CHAPITRE 2

Parallélisme et informatique distribuée

Introduction

Le parallélisme est un domaine de recherche au moins aussi vaste que le connexionnisme. Ce chapitre résume les notions indispensables à la compréhension des parallélisations du classifieur incrémental que nous étudierons dès le prochain chapitre. De nombreux ouvrages couvrent ce domaine de recherche, par exemple les livres de Quinn [Qui87] et de Cosnard *et al.* [CNR92].

L'Annexe B

Afin de faciliter la lecture, les notions et définitions indispensables sont présentées dans l'annexe B, notamment :

- la classification de Flynn (section B.1) ;
- la définition de la capacité d'évolution d'un système parallèle, de l'accélération, de la latence, de la bande passante, du taux de transfert, de la portabilité d'une solution parallèle, et d'un pipeline (section B.2) ;
- la notion de granularité (section B.2).

Le présent chapitre commence par présenter l'intérêt du mariage entre connexionnisme et parallélisme (sections 2.1 et 2.2). Mais l'utilisation d'une machine parallèle n'est pas l'unique solution au besoin de puissance du connexionnisme. Nous évaluons l'intérêt des processeurs dédiés au connexionnisme (section 2.3.1), des stations de travail haut de gamme (section 2.3.2), et des réseaux de stations (section 2.6). Cette dernière alternative aux machines parallèles sera mise en œuvre dans un prochain chapitre (section 5.1) grâce à un environnement de distribution du traitement nommé PVM. Le système PVM est décrit dans la section 2.7.

Parallélisation de modèles connexionnistes

Le chapitre se termine par une analyse des différentes manières de paralléliser un réseau de neurones sur l'architecture de machine choisie ou sur un réseau de stations (section 2.8). Les voies les plus intéressantes sont exploitées dans les prochains chapitres.

2.1 Parallélisme pour le modélisateur

Modéliser un système biologique recouvre deux préoccupations :

- comprendre une réalité biologique complexe en testant des systèmes artificiels plus simples, éprouver des hypothèses biologiques en simulant leur principe sur un ordinateur ;
- chercher l'inspiration d'une nouvelle approche algorithmique en observant le vivant, notamment pour des problèmes que l'humain résout facilement et qui sont délicats pour un ordinateur (par exemple lire un texte manuscrit ou reconnaître un visage).

Gain en rapidité d'exécution

Ces deux approches modélisatrices se heurtent à la lourdeur des réseaux connexionnistes : tester de nouvelles topologies suppose de nombreuses expériences [PM90, PM93]. Il faut trouver les paramètres intéressants pour le réseau (nombre de cellules et de couches, coefficients intervenant dans les règles d'apprentissage, *etc.*) et les faire varier dans leurs plages de valeurs respectives. De plus, les réseaux biologiques ont une taille largement supérieure à ce que nous autorise l'ordinateur.

Tâche de bas niveau

L'approche expérimentale n'est pas la seule raison du besoin en puissance de calcul. Un réseau de neurones fait ce que les informaticiens appellent un *travail de bas niveau*. Ce terme n'a rien de péjoratif, un travail de bas niveau désigne une tâche proche du matériel. Les fonctions de haut niveau, telles que lire une page de texte, sollicitent fortement le niveau inférieur. Aussi, un réseau de neurones artificiels doit être implémenté avec les techniques les plus rapides.

Le parallélisme pour le parallélisme

Enfin, au-delà des considérations pratiques, rappelons que le cerveau est fortement parallèle. Ce parallélisme intervient au niveau des cellules qui sont autant de systèmes indépendants. À un autre niveau de granularité, le cerveau est modulaire. De nombreuses expériences, notamment liées au processus d'apprentissage, peuvent mettre à profit les machines parallèles.

2.2 Parallélisme pour l'ingénieur connexionniste

L'ingénieur reprend à son compte les arguments développés pour le modélisateur, notamment la recherche de bons paramètres. Une raison supplémentaire de chercher la vitesse est l'objectif d'un traitement en « temps réel » (*real time processing* [HKS90]).

Définition

Un système dit « temps réel » est capable de répondre en un temps constant donné et imposé par l'application. On assimile souvent le temps réel à une réponse « immédiate ». En fait, selon le domaine d'utilisation du modèle connexionniste, cette contrainte de temps peut être de quelques dixièmes de secondes à plusieurs heures.

Exemples

Considérons l'exemple d'un réseau chargé de lire les codes postaux écrits sur des enveloppes défilant devant une caméra, le réseau doit déchiffrer les chiffres avant qu'une nouvelle enveloppe n'arrive sous la caméra. Prenons à présent l'exemple d'une application chargée d'établir les conditions météorologiques du lendemain. Une réponse en temps réel peut impliquer un calcul de plusieurs heures mais doit rester bien au-dessous des 24 heures.

Remarque

Notons que les réseaux à couches sont capables de donner une réponse en temps constant (ou du moins borné par une limite supérieure). Le temps de réponse dépend exclusivement du nombre de cellules, ou du nombre de couches si l'on considère un travail concurrent des cellules. En revanche, le temps constant est plus délicat à garantir pour un réseau récurrent, notamment si on attend que le réseau se stabilise sur une configuration. Il existe des théorèmes de convergence [Gol87], mais ils ne garantissent pas le nombre d'itérations nécessaires.

2.3 Pour aller plus vite

2.3.1 Circuit de neurones artificiels VLSI

Utilisation de « quelques transistors »

Nous venons de voir que le connexionnisme réclame une informatique rapide. Un moyen efficace pour effectuer un travail le plus rapidement possible est de faire un circuit VLSI (*Very Large Scale Integration*). Cette solution vient tout naturellement à l'esprit pour un réseau connexionniste. Il « suffit » de remplacer les cellules par quelques transistors, sur une carte ou dans un processeur, pour obtenir un réseau extrêmement véloce. Hélas, cette idée se heurte à au moins deux inconvénients majeurs.

1. L'architecture idéale n'est pas connue. On ne peut donc pas produire en grand nombre un circuit neuronal polyvalent.

2. Pour une architecture donnée, le dimensionnement du réseau dépend du problème. Il faut donc pouvoir choisir le nombre de couches et de cellules, ainsi que la répartition des cellules sur les différentes couches. La solution qu'apporte un modèle incrémental ne résout pas ce problème car un circuit est figé.

Ces problèmes peuvent être contournés en produisant des circuits implémentant des morceaux de réseau. Les briques de base sont ensuite connectées pour former le réseau désiré [RJHK89].

Apprentissage

Par ailleurs, un réseau doit être capable d'apprendre (section A.4), il faut donc ajouter à notre montage électronique les circuits nécessaires à la modification des poids de ses connexions. La surcharge est supportable pour des algorithmes simples, tels

que renforcer les connexions reliant les cellules actives simultanément. Cette stratégie d'apprentissage est d'inspiration biologique, elle est basée sur les travaux de Hebb [Heb49]. Cependant, les algorithmes les plus performants ont une inspiration plus mathématique que biologique (par exemple la minimisation d'une fonction de coût). Les circuits à mettre en œuvre pour intégrer de telles méthodes d'apprentissage sont lourds.

En conclusion, l'implantation de modèles connexionnistes sur des composants VLSI est possible et prometteuse, notamment pour des tâches de bas niveau (c'est-à-dire proche du matériel) [PA93, PF94]. Cependant, les volumes de production des processeurs généralistes qui équipent les stations de travail autorisent l'utilisation de technologies supérieures. Aussi, le rapport du coût et des performances ne permet pas encore aux réseaux de neurones artificiels « câblés » de sortir des laboratoires.

2.3.2 La « concurrence » du séquentiel

Vers un processeur généraliste universel

Comme nous venons de l'évoquer, les processeurs séquentiels actuels sont très efficaces. Ce résultat a une origine autant économique que scientifique. Des sommes considérables sont investies à chaque génération de processeurs, ce qui conduit à une concentration du marché autour de quelques processeurs généralistes. Cette production de masse de circuits puissants autorise la recherche des meilleurs compromis[1] entre fonctions et fréquences, ainsi que l'utilisation d'une technologie de pointe :

- le dessin est toujours plus fin, un nombre plus important de transistors se traduit par de nouvelles fonctions ;
- la fréquence augmente, les processeurs effectuent un plus grand nombre d'instructions par seconde sans pour autant surchauffer.

Parallélisme au sein du processeur

Au-delà des considérations technologiques, le processeur que nous qualifions de séquentiel tend à devenir une machine parallèle d'une très faible granularité. En effet :

- les processeurs travaillent sur plusieurs bits simultanément (16, 32, ou 64) ;
- des instructions SIMD font leur apparition [KB96, PWW97] ;
- les unités les plus sollicitées sont dédoublées ;
- des pipelines sont mis en œuvre, par exemple en décomposant « l'accès mémoire » ou le « décodage des instructions ».

1. On ne peut impunément ajouter des fonctions à un processeur car sa surface conditionne la probabilité qu'il contienne une impureté fatale à son fonctionnement. Augmenter la fréquence est également contraignant car le composant dissipe une importante chaleur qu'il faut évacuer sous peine d'instabilité (voire de destruction). Lorsqu'un ventilateur n'est plus suffisant, les coûts augmentent rapidement. Aussi, seules les fonctions les plus utilisées ont droit à un coin de processeur (on se passe par exemple souvent d'un circuit « diviseur »).

2.4 Choix d'une architecture parallèle

Choix d'une architecture MIMD

Dans l'état actuel de la recherche et de l'industrie, la supériorité des processeurs neuronaux et des machines SIMD sur les machines séquentielles n'est pas suffisante pour justifier les mêmes investissements humains et financiers. L'architecture MIMD tire avantage de la situation en intégrant les processeurs SISD généralistes du marché. C'est donc ce type de machine qui est la cible des parallélisations décrites dans cette thèse.

La classification de Flynn (voir l'annexe B) ne précise pas l'organisation des données. Aussi, la mémoire des machines MIMD peut être globale et partagée, ou locale et distribuée.

Mémoire partagée

Dans une machine à mémoire partagée, les processeurs se partagent dans le temps une même mémoire physique. Ces machines sont plus simples à programmer car aucun échange de messages n'est nécessaire. Cependant, de fortes contraintes technologiques, notamment le goulot d'étranglement de l'accès mémoire, limitent le nombre de processeurs.

Mémoire distribuée

En revanche, si chaque processeur a sa propre mémoire, à laquelle il peut seul accéder (figure 2.1), on parle de machine parallèle à mémoire distribuée. Lorsqu'un processeur a besoin d'une information située sur un autre processeur, des communications sont nécessaires. Le nombre important de communications que ce modèle peut engendrer impose une interconnexion performante - et coûteuse - des processeurs [Fen81, WF88].

Choix d'une mémoire distribuée

La nécessité d'une architecture capable d'évoluer en puissance (voir la section B.2.1) n'est pas respectée par l'utilisation d'une mémoire partagée puisque le nombre de processeurs est limité. Aussi, nous allons utiliser une machine parallèle MIMD à mémoire distribuée.

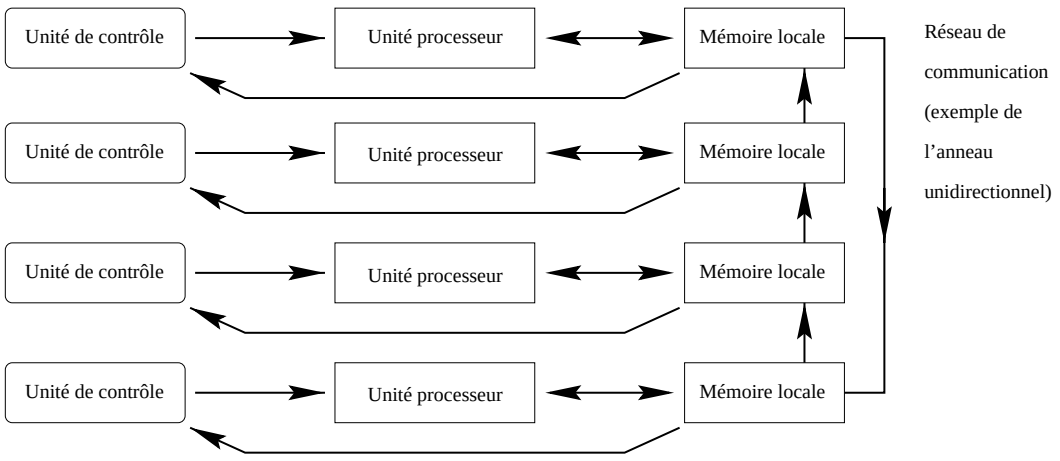

FIG. 2.1 – *Architecture parallèle MIMD*

Remarque sur les réseaux

Notons que l'interconnexion de plusieurs architectures SISD est une architecture MIMD à mémoire distribuée. Or, nous avons vu que les machines parallèles actuelles utilisent les mêmes processeurs que les stations de travail haut de gamme. Cette situation relativise l'intérêt d'une coûteuse machine parallèle face à un réseau d'ordinateurs séquentiels (nous tirerons parti de cette situation un peu plus loin, section 2.6).

Mémoire distribuée virtuellement partagée

Afin d'être complet, il est nécessaire de considérer l'utilisation d'une mémoire distribuée virtuellement partagée. Cette alternative à la mémoire distribuée est proposée pour la première fois par Li [Li86a, Li86b]. Le système met à la disposition du programmeur une machine virtuelle à mémoire partagée afin de facilité la programmation. Le système génère lui-même les communications nécessaires. Ainsi, la contrainte d'évolution introduite section B.2.1 est respectée. Cependant, cette stratégie peut avoir un important coût en temps d'exécution car la quantité de messages générés par le système est bien supérieure à ce qu'aurait produit un programme écrit directement pour mémoire distribuée. Cependant de récents développements apportent de nouvelles solutions [Lef97] (hiérarchisation, mélange des modèles de programmation, portabilité, *etc.*). Aussi, à la suite de cette thèse, nous entamons un portage de la version la plus efficace des parallélisations du classifieur sur une mémoire virtuellement partagée.

2.5 La Volvox IS-860

Nous avons utilisé une machine parallèle Volvox, de la société Archipel [Arc92], pour toutes les implémentations sur architecture MIMD à mémoire distribuée. Nous détaillons à présent les principales caractéristiques de cette machine.

Deux processeurs par nœud

La machine Volvox utilisée est une « IS-860 », elle comprend 48 « nœuds ». Chaque nœud est composé de deux processeurs :

- un processeur *i860* [Mar89] (de la société *intel*) dédié aux calculs ;
- un processeur *T805* [HMSS87] (de la société *inmos*) pour les communications.

Les deux processeurs « communiquent » entre eux via une mémoire partagée de 16 Mo, le *T805* dispose de 4 Mo supplémentaires. L'utilisateur peut utiliser les deux processeurs pour programmer la machine. La figure 2.2 schématise un nœud.

Portabilité et respect de l'architecture

Il est important de préserver la pérennité des parallélisations que nous allons proposer en ménageant leur portabilité (voir la section B.2.4). Aussi, afin de respecter au mieux la philosophie de l'architecture MIMD, tous les programmes que nous présentons pour la Volvox s'exécutent exclusivement sur les processeurs de calcul *i860*.

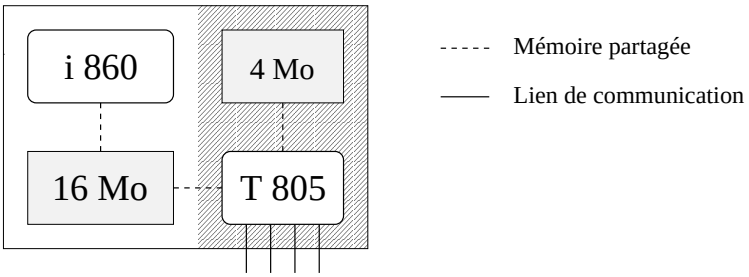

FIG. 2.2 – *Un nœud de la Volvox est composé de deux processeurs*

Les processeurs de communication *T805* sont laissés à la charge des bibliothèques de fonctions dédiées à l'acheminement des messages. De plus, les programmes et données de l'application sont stockés dans la mémoire accessible au *i860*. À la demande du processeur de calcul, le processeur de communication lit les messages à envoyer ou écrit les messages reçus dans cette même mémoire. Les 4 Mo stockent les programmes de gestion du réseau d'interconnexion et les données correspondantes (par exemple des tampons).
Le portage d'une application d'une architecture MIMD à une autre n'est jamais facile, cependant nos choix garantissent que ce portage est possible.

Topologie supportée

La Volvox est capable de connecter les nœuds pour réaliser une quelconque configuration choisie par l'utilisateur grâce à un « *crossbar* [Loo92, YYCYC93] », c'est-à-dire un « interconnecteur reconfigurable ». Le choix de la topologie est fait une fois pour toutes avant le lancement de l'application, grâce à un « script de configuration » relativement complexe écrit par l'utilisateur. Les chemins ainsi réalisés ne sont pas virtuels, un lien physique relie deux *T805* connectés. Aussi, la complexité des topologies que l'on peut mettre en œuvre sur une Volvox dépend directement du nombre de liens d'un *T805*. Comme on peut le remarquer sur la figure 2.2, un *T805* possède 4 liens, il peut donc être connecté à 4 autres *T805*. On dit que le « degré » maximal du graphe d'interconnexion est 4, cela inclut par exemple :

- l'anneau ;
- l'arbre binaire ;
- la grille et la grille torique ;
- l'hypercube (sous réserve d'une dimension maximale égale à 4).

Une machine parallèle multi-utilisateur

L'utilisation de liens physiques apporte une garantie sur la transmission, tant au niveau du temps d'initialisation que de la bande passante disponible. Un autre avantage est la possibilité qu'a le processeur de communication d'envoyer ou de recevoir simultanément des messages différents (sur des liens différents). Enfin, la machine est capable d'héberger plusieurs topologies disjointes simultanément, éventuellement

utilisées par plusieurs utilisateurs. Le nombre de personnes utilisant la machine est toutefois limité à 4 par le nombre de liens du *T805* installé sur l'ordinateur interfacé avec la Volvox (on parle d'ordinateur « hôte »), généralement une station de travail connectée à un réseau local.

Latence et taux de transfert de la Volvox

La machine étant équipée de *crossbars*, les valeurs de la latence (β) et du taux de transfert (τ) de processeur à processeur sont identiques quelle que soit la topologie (table 2.1). Un test complet du réseau de communication a été réalisé par Desprez, Gavoille, Jargot et Pourzandi [DGJP93].

La dernière ligne de la table ci-dessous concerne deux processeurs d'un même nœud. Les instructions de communication fournies par le constructeur de la machine traitent les messages échangés par les processeurs de calcul et de communication à un niveau logiciel. La lourdeur de la méthode explique le faible taux de transfert.

Heureusement pour notre stratégie de partage du travail (charge des communications aux *T805*), les *T805* peuvent accéder à la mémoire des *i860*, par exemple pour y écrire un message en cours de réception.

Intervenants	β (μs)	τ ($\mu s/o$)
Entre deux *T805*	766.5	1.10
Entre deux *i860*	1305.0	1.23
Entre un *T805* et un *i860*	1040.0	0.10

TAB. 2.1 – *Latence et taux de transfert de la Volvox*

Remarque sur les T805

Notons que les *T805* sont technologiquement un peu dépassés face aux *i860*. Une version de la Volvox équipée de *T9000* [BP93] devait voir le jour pour équilibrer les capacités de calcul et de communication de la machine. Mais la difficulté du marché a eu raison de la société Archipel à l'origine de la Volvox. Privée de support technique, la puissante machine est inutilisable. Cet ordinateur était pourtant d'une bonne qualité, les principaux problèmes rencontrés lors des parallélisations étant d'importants bogues dans les fonctions d'allocation mémoire. Bien que ce problème soit grave, nous avons pu le contourner dans nos implémentations en prenant systématiquement à notre charge toute la gestion dynamique de la mémoire. Un tel travail peut paraître rédhibitoire, cependant peu de machines parallèles sont simples à utiliser. Les outils de développement (compilateurs, débogueurs, scriptes de placement des programmes sur les processeurs, ...) sont le plus souvent très dépouillés et peu ou mal documentés. En conclusion, l'utilisation d'une machine parallèle est un travail difficile, même sans chercher à obtenir de bonnes performances. Aussi, les utilisateurs se sont tournés vers une alternative, notamment pour la phase de développement des applications : les réseaux de stations.

2.6 Machines parallèles contre stations de travail

Longévité des machines parallèles

Les machines parallèles sont performantes. Mais ces ordinateurs coûtent chers et les progrès constants de l'informatique rendent un modèle vite obsolète. Bien sûr le problème n'est pas nouveau et concerne aussi les ordinateurs séquentiels qui nous entourent. Cependant, il est facile de recycler un serveur d'applications en serveur de fichiers, puis en stations de travail, puis en station de saisie, et enfin en simple terminal connecté à un serveur d'applications de nouvelle génération. La difficulté est toute autre pour une machine parallèle : le manque de portabilité des applications (voire des algorithmes), la pauvreté des outils, et la complexité des machines, imposent de gros investissements humains.

Réseaux de stations de travail

La situation est toute autre sur un réseau de stations de travail. Or, nous avons vu que la concentration de l'industrie informatique conduit à trouver les mêmes processeurs dans les machines parallèles et dans les stations de travail, de plus les machines de bureau sont maintenant toutes en réseau. Cette configuration est compatible avec le modèle MIMD à mémoire distribuée.
Toutefois, un réseau local (typiquement un réseau « *Ethernet* ») ne peut rivaliser avec les moyens développés pour interconnecter les processeurs d'une machine parallèle. Les protocoles les plus récents (par exemple « *Fast Ethernet* » ou « ATM »), encore rarement utilisés, ne peuvent pas atteindre le niveau de performance d'un ordinateur MIMD. La machine parallèle a donc toujours sa place pour les applications de pointe.

2.7 Le système PVM

Présentation de PVM

Le système PVM (*Parallel Virtual Machine*) est un environnement logiciel permettant une distribution du travail sur un réseau de stations de travail [GBD+94, BDG+95]. Les premiers développements de PVM datent de 1989 au ORNL (*Oak Ridge National Laboratory*). L'installation du système est relativement simple et entièrement gratuite.

Hétérogénéité

Notons que l'environnement PVM supporte l'hétérogénéité à 3 niveaux : (i) matériel, plusieurs architectures peuvent cohabiter sur le réseau et se partager le travail ; (ii) logiciel, les machines peuvent fonctionner sous différents systèmes d'exploitation ; (iii) réseau, PVM gère de nombreux protocoles de réseaux locaux et globaux.

Définition du virtuel

Le mot « virtuel » est de plus en plus utilisé dans le langage courant, il est donc utile de préciser sa signification. Un dictionnaire associe le virtuel à l'éphémère et à l'irréel. Cette relation est probablement exacte pour les « particules virtuelles » des physiciens mais ne convient pas en informatique. Une ressource informatique virtuelle existe bel et bien, elle est simplement simulée pour prendre une forme

conventionnelle. Le virtuel est parfois assimilé au potentiel. La différence entre virtuel et potentiel est pourtant effective. Le potentiel existe seulement en puissance, c'est-à-dire dans le futur. Le virtuel est lui pleinement disponible à tout moment. En conclusion, une machine virtuelle n'a rien de précaire. Bien au contraire, la pérennité de la machine virtuelle est assurée par son indépendance face au « réel », et ainsi son fonctionnement sur de nombreuses machines réelles.

Principales architectures supportées

La table 2.2 donne une liste non-exhaustive des architectures supportées par PVM. Certaines de ces machines sont parallèles : la CM5 du constructeur *Thinking Machines*, et des machines iPSC 860, iPSC 2, et Paragon du constructeur *Intel*. Il ne faut pas considérer ces ordinateurs comme des nœuds d'une super machine virtuelle (bien que ce soit théoriquement possible) mais comme des machines parallèles utilisant PVM pour leur propre programmation dans un souci de portabilité des applications parallélisées. Cependant, quelques machines parallèles ne tolèrent que partiellement PVM, il est nécessaire d'utiliser des fonctions propriétaires (c'est-à-dire non PVM), notamment pour communiquer. Cette particularité autorise (sans forcément l'impliquer) de meilleures performances, au prix d'une perte de portabilité. De plus, la facilité d'installation de PVM annoncée peut être fortement diminuée.

Nouveaux développements Ordinateur de bureau

De nouveaux portages de PVM sont en cours. Les deux exemples suivants sont significatifs de l'amplitude de puissance supportée par PVM.

Une version présentée comme relativement stable de PVM pour *MS-Windows 95* et *MS-Windows NT* est disponible *via internet* depuis février 1997 [FD96]. Ce travail est significatif car la pénétration de ces systèmes est devenue majoritaire. De plus, si les performances graphiques et la fiabilité d'un ordinateur de bureau restent limitées, la puissance de calcul est de moins en moins ridicule.

Machines parallèles CAPITAN

Au sommet de la hiérarchie de puissance des ordinateurs actuellement sur le marché, on trouve par exemple la machine parallèle MIMD à mémoire distribuée CAPITAN, de la société MATRA. La machine CAPITAN est livrée avec un environnement de programmation efficace mais propriétaire nommé CAPCASE. Une surcouche logicielle, réalisée par Prylli [Pry95], implémente les bibliothèques de l'environnement PVM. La diminution des performances est très faible. Garder l'environnement propriétaire met à profit le travail de qualité déjà réalisé sur CAPCASE, ainsi que les évolutions à venir.

Simulation d'une machine MIMD

Bien que PVM supporte des machines très diverses, il est souhaitable de se limiter à un réseau de machines de puissance équivalente (c'est-à-dire ne pas mêler un *Cray* et une *Sparc 1*) pour simuler une machine parallèle de type MIMD à mémoire distribuée. De plus, il est préférable d'utiliser un réseau disposant d'une faible latence et d'une bonne bande passante. Cela exclut *a priori* des machines connectées sur un réseau global (par exemple *internet*).

Architecture	Conditions éventuelles de fonctionnement
Alliant FX8	
DEC Alpha	Sous système DEC OSF-1
Sequent Balance	Sous système DYNIX
BBN Butterfly TC2000	
PC 80386 et 486 sous Unix	Sous systèmes BSDI, 386BSD, ou NetBSD
Thinking Machines CM2	Machine connectée à une station Sun
Thinking Machines CM5	
Convex C-series	
Cray C-90, YMP, T3D	Sous système UNICOS
Cray-2 et Cray S-MP	
HP-9000 modèle 300 et PA-RISC	Sous système HPUX
Intel iPSC 860	
Intel iPSC/2	Machine connectée à un 386, sous système V
PC 80386 et 486 PC sous LINUX	
Maspar	
MIPS 4680	
NeXT	
Intel Paragon	
DECstation 3100, 5100	Sous système Ultrix
IBM RS6000 et IBM 370	Sous système AIX 3.2
IBM RT	
Silicon Graphics IRIS	Sous systèmes IRIX 4.x ou IRIX 5.x
Station Sparc	Sous systèmes SunOS 4.2 ou Solaris 2.x
Station Sparc multiprocesseurs	Sous système Solaris 2.x (mémoire partagée)
DEC MicroVAX	

TAB. 2.2 – *Architectures et systèmes d'exploitation supportant PVM*

Mesure du temps sous PVM

L'utilisation de PVM ne présente pas que des avantages. Si l'environnement de programmation disponible sur une machine parallèle est moins confortable et moins portable que PVM, certaines fonctions proches du niveau matériel font défaut à PVM. Ainsi, il est difficile (voire impossible) de calculer une accélération sous PVM :

- l'environnement PVM ne donne pas accès à un temps précis et global ;
- synchroniser les temps locaux est délicat puisque des communications sont nécessaires sur un réseau utilisé par d'autres utilisateurs.

Démarrage virtuel

De plus, l'initialisation de la machine virtuelle passe par le lancement d'un certain nombre de services - de processus - sur chaque machine du réseau participant à la

configuration virtuelle. Cependant, cette initialisation est comparable au démarrage d'une machine parallèle. Ce temps ne dépend pas de la taille du problème à traiter. En conséquence, pour un problème connexionniste, l'initialisation de PVM est négligeable à partir d'une certaine taille de la base d'exemples à apprendre ou à généraliser.

2.8 PVM en perspective

Nous avons vu que PVM permet de simuler une machine parallèle. De façon plus générale, PVM rend possible une distribution du traitement sur un réseau de machines. Comme pour une machine parallèle, le but peut être d'accélérer au maximum une application pour répondre à un besoin précis. Nous allons maintenant voir une autre utilisation intéressante de PVM.

Gros système et terminaux

Un gros système, schématisé figure 2.3 centralise tous les traitements sur une seule machine très puissante à l'architecture SISD, éventuellement une machine multiprocesseurs (type SMP). Les utilisateurs interagissent avec le système *via* des terminaux dépourvus de puissance de calcul[2]. Moins le nombre d'utilisateurs est important, ou moins chaque utilisateur présent sollicite la machine, plus les applications disposent de ressources (processeur(s), mémoire, disques durs, *etc.*). Par ailleurs, la centralisation facilite le travail de l'administrateur du système qui peut aisément assurer la sécurité du système. Cependant, ces systèmes représentent de gros investissements et peuvent difficilement évoluer. Lorsque la puissance du système est rattrapée par la gourmandise des environnements et des applications, ou par le nombre d'utilisateurs, le coût de la mutation est élevé. Enfin, un gros système utilise généralement une architecture dite « propriétaire » incompatible d'un système à un autre. Notons que nous avons déjà fait les mêmes critiques à l'encontre des machines parallèles.

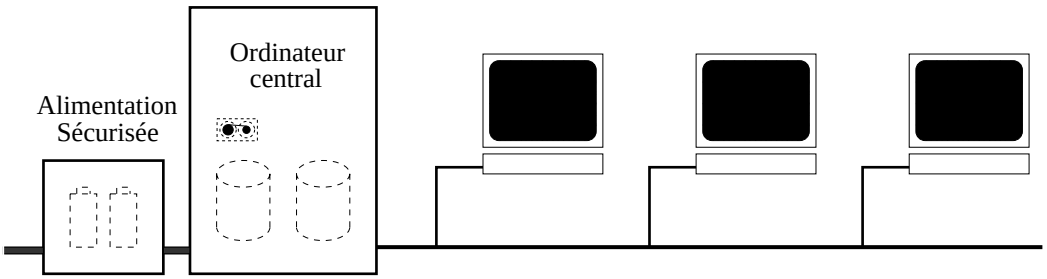

FIG. 2.3 – *Un ordinateur central puissant et des terminaux*

2. Avec le temps, ces terminaux ont tout de même gagné en « intelligence », passant par exemple d'une simple « ligne de commande » à un support actif d'environnements graphiques (terminaux « X »). Mais les processeurs éventuellement présents ne sont pas utilisés pour les applications.

Réseau de stations et terminaux

Contrairement à un terminal, une station de travail possède un processeur et de la mémoire tout en restant d'un coût modéré. Ces machines sont souvent équipées d'un disque dur ou d'autres périphériques, en fonction de leur utilisation. Bien qu'une station de travail puisse être autonome, ces machines sont conçues pour fonctionner en réseau (la figure 2.4 donne un exemple). Un utilisateur peut alors utiliser les ressources d'une autre machine, ce qui évite la prolifération des équipements les moins utilisés ou onéreux. Certains équipements comme les disques durs sont souvent partagés, en restant distribués dans les machines ou en étant regroupés dans un « serveur de fichiers ». Notons qu'un utilisateur peut se servir de sa station (ou d'un terminal) pour se connecter sur une autre station ou sur un gros système. Bien que l'architecture des machines soit propriétaire, de nombreux protocoles sont ouverts (le contraire de « propriétaire »), notamment pour le réseau et l'affichage. Il est donc possible de construire des réseaux hétérogènes. Enfin, les réseaux sont le plus souvent interconnectés (par exemple *via internet*). À titre d'exemple, il est possible pour un utilisateur de lancer un traitement sur une machine distante et de demander à ce que cette machine utilise l'écran de la station locale.

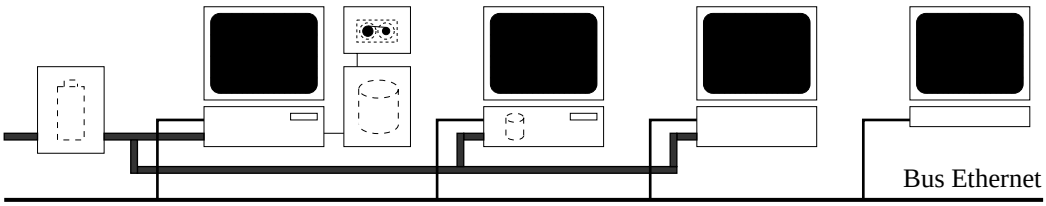

FIG. 2.4 – *Exemple d'un réseau de stations (avec un terminal)*

Réseau d'ordinateurs de bureau

Parallèlement aux stations de travail, l'ordinateur personnel de type « IBM PC », peu puissant mais simple et économique, a fait son apparition sur les bureaux. La polyvalence de ces petits systèmes, ainsi que leur architecture matérielle ouverte, en a généralisé l'utilisation dans les entreprises et chez les particuliers et a permis d'atteindre d'excellents rapports puissance / coût. L'absence de système d'exploitation de qualité a longtemps rendu l'administration de ces systèmes laborieuse. La mise en réseau de ces machines et l'apparition récente de systèmes d'exploitation performants (*Linux*, *Nextstep*, *Solaris*, *etc.*) a fortement diminué l'écart entre ces machines et le monde des stations de travail.

Un potentiel inexploitable

Mais les réseaux de machine n'ont pas que des avantages. En effet, si l'utilisateur peut utiliser toutes les ressources des machines auxquelles il a accès, la démarche est délicate. On se contente donc généralement de mettre à profit une ou deux machines du réseau pour y lancer ponctuellement des tâches. Le choix des machines est manuel, il s'agit généralement d'une machine que l'on sait peu sollicitée car non utilisée ou faiblement chargée (éditeur de texte, courrier électronique, *etc.*).

Exploitation du potentiel via PVM

Un environnement comme PVM est une réponse à cette limite. En effet, PVM offre l'opportunité d'utiliser les ressources disponibles. Dans ces conditions, atteindre une accélération linéaire n'est plus une priorité. L'objectif est d'accélérer le traitement sans pour autant gêner les autres utilisateurs. Il est intéressant de remarquer que si chaque utilisateur distribue ainsi son travail, le réseau de stations (local ou global) peut être vu comme une seule puissante machine virtuelle (un « *mainframe* virtuel »).

Conclusion

Une bonne maîtrise du parallélisme est indispensable mais non suffisante pour la mise au point d'un programme parallèle efficace : une stratégie de parallélisation est nécessaire. Il existe de nombreuses manières de paralléliser une application sur une machine MIMD à mémoire distribuée, et plus particulièrement un réseau de neurones.

Méthode de l'espion

Si l'objectif de la parallélisation est uniquement la recherche d'une bonne topologie (nombre de couches, nombre de cellules sur chaque couche, ...) ou de bons paramètres pour le réseau (par exemple $\Theta_{influence}$ et $\Theta_{confusion}$ dans le cas du classifieur incrémental) il n'est pas indispensable de paralléliser le modèle connexionniste en lui-même. La méthode dite « de l'espion [PM90, PM93, BPM94] » s'acquitte efficacement de cette tâche en confiant à chaque processeur de la machine parallèle (de type MIMD à mémoire distribuée) un même réseau de neurones, mais avec des paramètres différents. L'objectif est de déterminer un jeu de paramètres adapté à une application donnée. Dès qu'un processeur est disponible, l'espion lance de nouveaux réseaux en prenant en compte les résultats déjà collectés pour couvrir intelligemment l'espace des paramètres.

Partage du réseau

Si la mise en œuvre d'un espion n'est pas suffisante (c'est-à-dire si le besoin de vitesse dépasse le stade de la mise au point du réseau pour une application donnée) une solution naturelle consiste à partager le réseau entre les processeurs (figure 2.5). Cependant, la granularité des machines MIMD n'autorise pas de pousser cette stratégie à l'extrême en utilisant un processeur par cellule (voir si nécessaire l'annexe, section B.2.6).

Exemple du partage d'un réseau à couches

On peut par exemple paralléliser un réseau à couches en confiant une couche à chaque processeur, une fois toutes les cellules d'une couche activées, une communication propage l'influx vers la couche suivante [PPDG89]. Si les couches sont équilibrées il est possible de mettre en œuvre un pipeline sur les exemples et de recouvrir les temps de calcul et de communication [PDG93]. La méthode est simple en généralisation mais se complique en apprentissage. De plus, le nombre de processeurs utilisables dépend du nombre de couches. Même en incluant le prétraitement, le nombre de niveaux du pipeline (et donc l'accélération) sera faible.

Cas du classifieur incrémental

Une parallélisation du classifieur incrémental par partage du réseau, avec utilisation d'un pipeline, est l'objet du prochain chapitre. Chaque processeur traite un morceau de la forme en entrée.

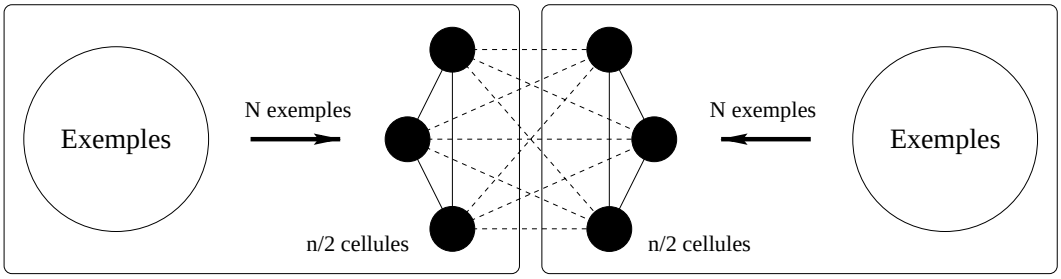

FIG. 2.5 – *Principe du partage du réseau*

Distribution des exemples

Mais le partage du réseau n'est pas la seule solution, une autre méthode de parallélisation consiste à placer un réseau sur chaque processeur, et à distribuer les exemples (figure 2.6).

Exemple pour un réseau à couches

On trouve cette méthode utilisée avec succès pour des réseaux à couches dans la littérature du connexionnisme et du parallélisme. La difficulté vient de la nécessité de maintenir une cohérence entre les poids de chaque copie du réseau, *via* des communications. Il faut mettre en œuvre une stratégie de mise à jour des poids élaborée pour atteindre de bonnes performances [GPM93, GPM94, Gir94]. La solution réside en un compromis entre la fréquence des communications, et le nombre d'exemples nécessaires pour atteindre un taux de succès en généralisation donné.

Cas du classifieur incrémental

Une parallélisation du classifieur incrémental par distribution des exemples fait l'objet des chapitres 4 et 5. Nous rencontrerons les mêmes difficultés que pour les réseaux à couches, notamment pour la gestion de la charge de communications nécessaires à la mise à jour des poids. Cependant, nous verrons qu'une « stratégie d'apprentissage » permet d'atteinde des performances bien plus satisfaisantes dans le cas du classifieur incrémental.

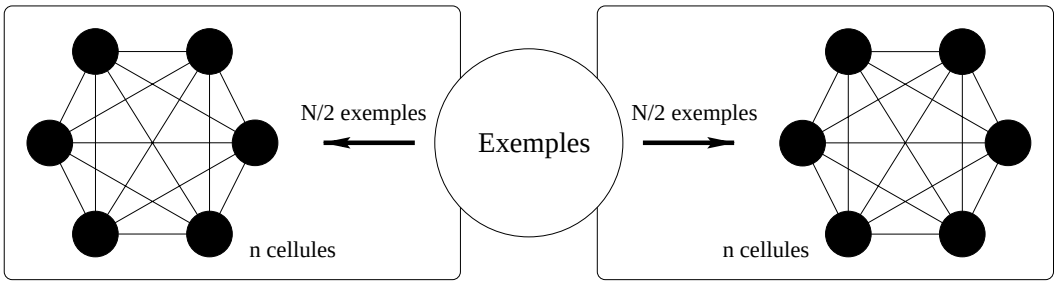

FIG. 2.6 – *Principe de la distribution des exemples*