HAL
archives-ouvertes.fr

Un outil de développement parallèle des réseaux de neurone

Yann Boniface

▶ **To cite this version:**

Yann Boniface. Un outil de développement parallèle des réseaux de neurone. Neurosciences et Sciences pour l'Ingénieur, Sep 2000, Dinard, France, 4 p, 2000. <inria-00099048>

HAL Id: inria-00099048
https://hal.inria.fr/inria-00099048

Submitted on 26 Sep 2006

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Un outil de développement parallèle des réseaux de neurones

Yann Boniface

LORIA - INRIA Lorraine, Campus scientifique B.P. 239 F-54506 Vandœuvre-lès-Nancy Cedex
E-mail : boniface@loria.fr

1 Les objectifs

Notre recherche a pour but l'étude du parallélisme intrinsèque des réseaux de neurones à inspiration biologique, qui sont des modèles à calculs totalement distribués. Nos objectifs sont doubles. Tout d'abord, nous souhaitons pouvoir utiliser cette propriété dans l'étude et le développement de nouveaux modèles neuromimétiques. Ensuite nous voulons exploiter ce parallélisme *neuronal* pour implanter nos modèles sur machines parallèles MIMD à mémoire partagée [4]. L'un des problèmes liés à l'utilisation des réseaux de neurones est en effet le fort coût de ceux-ci en temps de calcul, et ce essentiellement en phase d'apprentissage. Il est donc intéressant de pouvoir utiliser la puissance de calcul des machines parallèles modernes pour accélérer l'exécution de nos réseaux, pour les configurer comme pour les exécuter [7]. L'utilisation de la machine parallèle peut aussi nous permettre de construire des réseaux plus complexes et de travailler sur de plus vastes bases de données.

2 Méthode

S'il existe de nombreux travaux portant sur l'implantation parallèle des réseaux connexionnistes [7], il s'agit le plus souvent d'adaptation des algorithmes aux machines massivement parallèles. Ces applications, si elles accélèrent les exécutions, nécessitent des compétences en programmation parallèle pour être manipulées, ce qui, comme outil de recherche en connexionnisme, n'est pas satisfaisant. Les modèles de simulation de réseaux neuromimétiques sur machines parallèles sont beaucoup plus rares et ils limitent souvent les implantations aux modèles les plus classiques (réseaux à couches, réseaux non récurrents, cartes auto-organisatrices, etc.) [6] et/ou proposent des langages de programmation spécifiques, ce qui les rends peu attractifs du fait de l'indispensable apprentissage de ces langages [5].

Nous souhaitons, pour notre part, développer un outil permettant l'utilisation des machines parallèles pour accélérer les exécutions, tout en offrant aux connexionnistes une facilité d'implantation de leurs réseaux, en utilisant les propriétés parallèles propres aux modèles neuromimétiques. Cet outil doit donc permettre d'implanter les réseaux les plus classiques comme les réseaux plus expérimentaux, l'implantation doit pouvoir être relativement rapide et aisée et, utilisation de machine parallèle oblige, l'accélération de l'exécution des applications doit être "significative". De plus, pour faciliter le développement des réseaux de type biologique, nous proposons de résoudre certains des problèmes récurrents du domaine (validité des informations, synchronisations des différents neurones, communications entre les neurones, création de neurones, etc.).
Pour approcher ces objectifs, nous avons choisi de développer une bibliothèque permettant d'implanter les réseaux de neurones comme une somme d'objets distribués, autonomes et synchronisés par cycles. Cette bibliothèque est un ensemble de quelques fonctions, sur le langage

"C", permettant de définir des neurones, des couches de neurones, des réseaux de neurones, et d'exécuter ces réseaux sur machines séquentielles classiques comme sur les machines parallèles MIMD à mémoire partagée. Il semble en effet utile de pouvoir tester et construire les modèles sur station de travail classique et de garder les machines parallèles pour les calculs lourds.

3 Vue générale de l'implantation d'un réseau

Du point de vue de l'utilisateur, un réseau de neurones s'implante en définissant chaque type de neurone du réseau comme un objet, tel que décrit dans la figure 1. Pour la bibliothèque, un neurone est vu comme un objet disposant d'une sortie unique, d'un certain nombre d'entrées et de variables locales. Une entrée correspond à un canal de connexion à la sortie d'un autre neurone du réseau, à travers lequel le neurone peut lire la valeur de la sortie du neurone auquel il est connecté (au cours d'un cycle *t*, c'est l'activité du neurone "cible" au cours du cycle *t-1* qui est lue). Les connexions d'un neurone peuvent se faire de manière dynamique i.e un neurone peut créer ou éliminer des connexions à tout moment de sa vie dans le réseau.

Pour décrire le comportement du réseau le programmeur doit spécifier la tâche que doit accomplir chaque neurone au cours de chacun de ses cycles de vie. Cette spécification se fait à l'aide des trois fonctions, dites *fonctions caractéristiques* du neurone, suivantes :

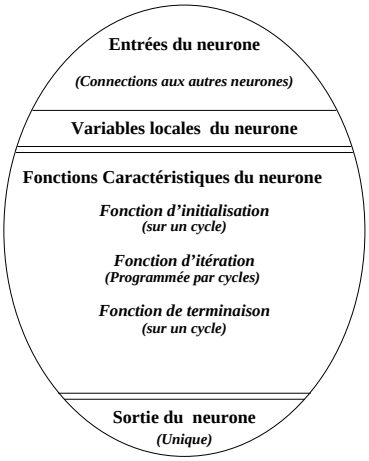

FIG. 1 – *Description d'un neurone*

- **Fonction d'initialisation**
  Elle est exécuté au premier cycle de vie du réseau. Elle permet, en général, au neurone de définir ses différentes connexions, ses variables locales et sa sortie.
- **Fonction d'itération**
  Elle est exécutée du second au pénultième cycle de vie du neurone. Elle décrit, en général, la méthode de calcul de la sortie du neurone, en fonction de ses entrées (au travers de ses connexions) et de ses variables locales.
- **Fonction de terminaison**
  Elle est exécutée au dernier cycle de vie du neurone.

L'exécution d'un réseau se termine à la mort de tous les neurones qui le composent. Un neurone peut mourir à tout moment de l'exécution du réseau. De la même façon, tout neurone peut demander la création d'un nouveau neurone dans le réseau. Notre bibliothèque fournit, de plus, des outils permettant de gérer les neurones par couches, au niveau de leur création, de leur destruction et de leurs connexions.

Ainsi défini, cet outil permet donc de construire les grands types de réseaux connexionnistes classiques, réseaux à couches, réseaux incrémentaux et réseaux plus "biologiques" de manière totalement distribuée. Il permet aussi l'utilisation des principales méthodes d'apprentissage (hebb, rétropropagation du gradient, méthodes d'élagage, etc.).

La bibliothèque assure, pour l'utilisateur, le synchronisme, la communication et la cohérence des données entre les différents neurones du réseau. En exécution sur machine parallèle,

elle gère, de manière totalement transparente, la distribution des neurones sur les différents processeurs, les problèmes de placement de mémoire, de gestion des caches, le synchronisme entre les neurones et entre les processeurs.
Des descriptions plus précises des implantations avec la bibliothèque, et des aspects techniques de celle-ci, peuvent être trouvé dans les articles [1] et [2].
Nous avons récemment enrichi notre bibliothèque d'outils graphiques permettant de visualiser l'évolution de nos réseaux. Sur machines parallèles, ces outils permettent la visualisation de l'exécution sans perte des gains de temps induits par l'utilisation d'ordinateur parallèle MIMD.

4 Résultats

Après avoir implanté divers réseaux, nous avons pu vérifier que notre bibliothèque facilite l'écriture des réseaux de neurones. Nous avons, en particulier, développé un algorithme d'élagage d'un OWE. Dans ce réseau de type perceptron multi-couches, chaque poids est déterminé par un perceptron multi-couches propre, l'apprentissage se faisant par rétropropagation du gradient [3]. L'utilisation de notre bibliothèque se fait à l'aide des algorithmes séquentiels classiques, le passage sur ordinateurs parallèles ne nécessite donc aucune connaissance spécifique de ces machines et de leur programmation de la part du programmeur.

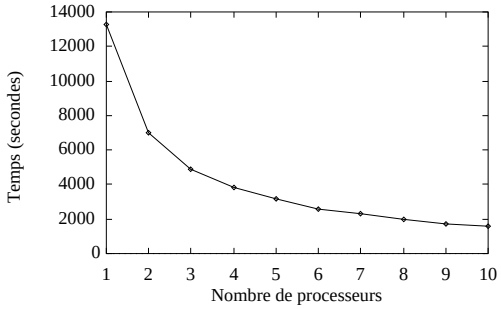

FIG. 2 – *Temps d'exécution d'un réseau de type growing neural gas en fonction du nombre de processeurs utilisés*

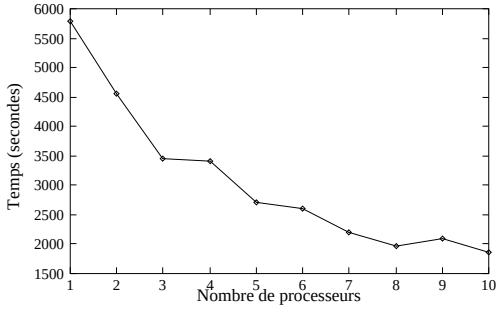

FIG. 3 – *Temps d'exécution d'un réseau de type OWE en fonction du nombre de processeurs utilisés*

Au niveau des performances sur machines parallèles, et pour des réseaux de grande taille, nous obtenons de bonnes accélérations sur des réseaux de neurones *biologiquement plausibles*

(Carte de Kohonen ou Growing Neural Gas) sur huit à dix processeurs (voir figure 2). Sur des modèles plus "mathématiques", comme les Perceptrons Multi Couches à rétro-propagations du gradient, nos performances sont plus mesurées (3 fois plus rapide sur 6/8 processeurs) comme le montre la courbe de la figure 3 obtenue par l'exécution d'un réseau OWE [3], réseau à couches avec apprentissage par rétropropagation du gradient. Ces résultats sont obtenus sur une SGI-Origin2000, ordinateur parallèle MIMD à mémoire partagée.

Plusieurs facteurs peuvent expliquer ces résultats. Tout d'abord, notre bibliothèque est généraliste : elle a l'ambition de permettre d'implanter tous les types de réseaux connexionnistes, elle n'est donc pas optimisée pour un modéle spécifique ou pour un algorithme d'apprentissage précis. Ensuite notre outil est construit pour utiliser le parallélisme intrinsèque au modèles biologiques et l'adapter aux machines MIMD. Or les modèles de type Perceptron Multi-Couches ne sont pas biologiquement plausibles. Leur algorithme d'apprentissage (rétro-propagation du gradient) pourrait se décrire comme *séquentiel par couches* : seuls quelques neurones du réseau sont actifs simultanément. Enfin, les neurones sont des unités effectuant, au cours de chaque cycle, un faible calcul pour déterminer leur activité, à partir de leurs entrées, avant de diffuser celle-ci aux autres neurones. Ceci entraîne une fréquence élevée des communications et des synchronisations entre neurones, et donc de nombreuses communications et synchronisations entre les différents processeurs, ce qui limite les performances parallèles.

5 Conclusion

Nous avons développé une plate-forme, sous forme de bibliothèque, d'implantation de réseaux de neurones permettant des exécutions sur ordinateurs séquentiels classiques comme sur machines massivement parallèles (type MIMD à mémoire partagée). Cette bibliothèque propose d'utiliser les propriétés de parallélisme à grain fin des réseaux connexionnistes de deux manières distinctes. D'abord pour implanter les réseaux sous une forme totalement distribuée. En utilisant ensuite ce parallélisme pour adapter les réseaux au parallélisme à gros grain des machines parallèles MIMD, ce qui permet de masquer le parallélisme matériel et de rendre attractif l'utilisation de ces machines pour les programmeurs de réseaux connexionnistes.

L'aspect générique de notre plate-forme, obtenu par l'utilisation de la mémoire partagée, permet d'implanter tous les types de réseaux. En revanche, cet aspect générique, en raison des nombreuses communications et synchronisations inhérentes au modèle, limite l'obtention de performances à des exécutions sur peu de processeurs en exécution parallèle (une dizaine).

Références

1. Y. Boniface, F. Alexandre, and S. Vialle. A bridge between two paradigms for parallelism : Neural networks and general purpose mimd computers. In *IJCNN*, 1999.
2. Y Boniface, F Alexandre, and S Vialle. A library to implement neural networks on MIMD machines. In *EuroPar'99*, 1999.
3. L Bougrain and Y Boniface. Détermination de l'architecture de modèles neuromimétiques par implémentation parallèle. In *Journée Scientifiques du CCH*, 2000.
4. H. Paugam-Moisy. Multiprocessor simulation of neural networks. In *The Handbook of Brain Theory and Neural Network.*, pages 605–608. The MIT Press., 1995.
5. L. Precheld. Cupit-a parallel language for neural algorithms : Language reference and tutorial. *Tecnical Report, Univ. Karlsruhe, Allemagne*, 1994.
6. Thilo Reski. *Mapping and Parallel, Distributed Simulation of Neural Networks on Message Passing Multiprocessors*. PhD thesis, Universität Paderborn, 1999.
7. S. Shams and J.L. Gaudiot. Parallel implementations of neural networks. *International Journal on Artificial Intelligence Tools*, 2(4):557–581, 1993.