École Doctorale IAEM Lorraine **École Nationale d'Ingénieurs de Sousse**

Réalisation d'un Réseau de Neurones *"SOM"* Sur une Architecture Matérielle Adaptable et Extensible à Base de réseaux sur Puce *"NoC"*

THÈSE

présentée et soutenue publiquement le 7 Juillet 2018

pour l'obtention du

Doctorat de l'Université de Lorraine
en cotutelle avec l'Université de Sousse

(Mention Systèmes Électroniques (IAEM)
Mention Génie Electrique (ENISo))

Auteur:

Mehdi ABADI

Composition du jury

Présidents :	Mme. Najoua ESSOUKRI BEN AMARA, Professeur, *LATIS*, Université de Sousse, Tunisie
Rapporteurs :	Mme. Fan YANG, Professeur, *LE2I*, Université de Bourgogne, France M. Adel GHAZEL, Professeur, GRESCOM, Université de Carthage, Tunisie
Examinateurs :	M. Ali DOUIK, Professeur, ENISo, Université de Sousse, Tunisie
Directeurs de Thèse :	M. Mohamed Hédi BEDOUI, Professeur, TIM, Université de Monastir, Tunisie M. Serge WEBER, Professeur, IJL, Université de Lorraine, France
Co-directeurs de Thèse :	M. Slaviša JOVANOVIĆ, MDC, IJL, Université de Lorraine, France M. Khaled BEN KHALIFA, MDC, TIM, Université de Monastir, Tunisie

Institut Jean Lamour (UMR7198)

Laboratoire de Technologie et Imagerie Médicale (LR12ES06)

Remerciements

J'adresse mes sincères remerciements à Madame *Fan YANG*, Professeur à l'Université de Bourgogne ainsi qu'à Monsieur *Adel GHAZEL*, Professeur à l'Université de Carthage et directeur du laboratoire de recherche *GRESCOM* à l'École Supérieure des Communications de Tunis, qui m'ont fait l'honneur de juger cette thèse en qualité de rapporteurs.

Je tiens également à remercier Madame *Najoua ESSOUKRI BEN AMARA*, Professeur à L'Université de Sousse et directrice du laboratoire de recherche LATIS à l'École Nationale d'Ingénieurs de Sousse et Monsieur *Ali DOUIK*, Professeur à l'Université de Sousse, d'avoir accepté d'examiner ce travail et de participer à ce jury.

J'adresse mes remerciements à Monsieur *Mohamed Hedi BEDOUI* Professeur à l'Université de Monastir et directeur du Laboratoire LTIM à la Faculté de Médecine de Monastir, de m'avoir accueilli au sein de son laboratoire de recherche. Je tiens également à remercier Monsieur *Serge WEBER*, Professeur à l'Université de Lorraine de m'avoir accueilli au sein du Laboratoire IJL, Nancy. Je les remercie plus particulièrement, pour la confiance qu'ils ont su m'accorder tout au long de cette thèse et pour le fait d'avoir accepté d'être mes directeurs de thèse.

Je voudrais exprimer ma gratitude à Monsieur *Slaviša JOVANOVIĆ*, Maître de Conférences à l'Université de Lorraine. Je le remercie de m'avoir encadré durant cette thèse, pour ses qualités et sa rigueur scientifique, sa disponibilité et son soutien permanent qui m'ont beaucoup aidé tout au long de ce travail.

Je tiens également à remercier Monsieur *Khaled Ben Khalifa*, Maître de Conférences à l'Institut Supérieur des Sciences Appliquées et de Technologie, de m'avoir encadré durant cette thèse. Je le remercie plus particulièrement pour ses encouragements, pour ses précieux conseils scientifiques et humains et pour son soutien moral tout au long de cette thèse.

Mes remerciements les plus chaleureux à tous les membres du laboratoire TIM ainsi que les membres de l'équipe 406 de l'IJL pour leur aide ou les conseils qu'ils ont pu m'apporter, tout particulièrement à Madame *Amina AMMAR* et Madame *Christine DAUDENS* pour leurs conseils administratifs.

Je tiens à remercier tous mes amis, tout particulièrement *Ahmed, Houssem, Zied, Iskander, Achraf et Ahmed Ghazi* pour leur soutien et leurs questions « répétitives » concernant mon sujet de travail et son utilité finale.

Mes remerciements sincères vont également à ma famille, en particulier à ma mère, mon frère *Salem* et son épouse *Hadil*, ma sœur *Nouha* et son mari *Saif* et ma petite *Abir*. Je vous remercie du fond du cœur pour votre compréhension, votre confiance et votre soutien.

Afin d'être reconnaissant envers ceux qui m'ont appuyé et encouragé à effectuer cette thèse, je remercie tous ceux qui ont contribué à ce travail de recherche, qu'ils trouvent ici l'expression de toute ma reconnaissance.

Mehdi ABADI

Avril 2018

"La vie, c'est comme une bicyclette, il faut avancer pour ne pas perdre l'équilibre"
Albert Einstein

Je dédie cette thèse
à l'âme de Mon cher Père,
à ma Ma chère mère,
Du fond de mon cœur.

Résumé

Depuis son introduction en *1982*, la carte auto-organisatrice de *Kohonen* (*Self-Organizing Map : SOM*) est devenue l'un des outils connexionnistes les plus utilisés. Ce modèle neuronal a prouvé ses capacités de classification et visualisation des données multidimensionnelles dans différents domaines.

Dans la littérature, il existe plusieurs travaux d'implémentation de la carte *SOM* avec différentes approches : matérielle, logicielle ou mixte. Les implémentations matérielles, en exploitant le taux de parallélisme élevé de l'algorithme de *Kohonen*, permettent d'augmenter les performances de ce modèle neuronal souvent au détriment de la flexibilité. D'autre part, la flexibilité est offerte par les implémentations logicielles qui quant à elles ne sont pas adaptées pour les applications temps réel à cause de leurs performances temporelles limitées. Certaines applications temps réel, avec des contraintes temporelles fortes, favorisent l'utilisation des implémentations matérielles, tout en gardant un certain niveau de flexibilité permettant d'améliorer la précision et la qualité des résultats attendus.

Cela nous a amené à proposer une architecture matérielle de la carte SOM dynamiquement reconfigurable permettant de garantir un certain niveau de flexibilité. Dans ce cadre, nous avons changé la boite d'outil souvent utilisée pour la conception des architectures matérielles des réseaux de Kohonen, en y ajoutant une technique de communication de type réseaux sur puce (*Network on Chip : NoC*). Cette technique permet d'éviter les limites de flexibilité et extensibilité causées par les communications point à point souvent utilisées. D'autre part, le traitement de l'algorithme de Kohonen a été totalement distribué pour éviter la dépendance architecturale au comportement global de la carte *SOM*.

A base de cette approche, nous avons mis en œuvre une architecture matérielle adaptable et extensible de la carte *SOM* dédiée pour une implantation sur *FPGA*. Cette architecture permet de reconfigurer, par un simple paramétrage, la structure de la carte *SOM* (le nombre de neurones, leur topologie de distribution et la dimension du vecteur de poids) selon les besoins spécifiques à une application, sans refaire à nouveau la conception de l'architecture. D'autre part, nous avons également proposé une architecture matérielle innovante d'une carte *SOM* à structure croissante au cours de la phase d'apprentissage.

Mots-clés: Réseau de neurones artificiels (RNA), Carte auto-organisatrice (SOM), Réseau sur puce (NoC), Multiprocesseurs système sur puce (MPSoC), Architecture auto-adaptable, Réseau de neurones évolutif

Abstract

Since its introduction in *1982*, Kohonen's *self-organizing map (SOM)* became one of the most used connectionist tools. This neuronal network proved its ability to classify and visualize multidimensional data in various fields of application.

In the literature, there are several research works about the SOM implementation by using different approaches : hardware, software or co-designed. The hardware implementations, by exploiting the inherent parallelism of the Kohonen algorithm, allow to improve the overall performances of this neuronal network often at the expense of the flexibility. On the other hand, the flexibility is offered by software implementations, which are on their side not suited for real-time applications due to the limited performances. Some real-time applications with hard time constraints, prefer the use of hardware implementations, while maintaining a limited flexibility to improve the accuracy and the quality of the expected results.

Therefore, in this PhD thesis we proposed a hardware dynamically reconfigurable *SOM* architecture guaranteeing a high level of flexibility. In this context, we have changed the toolbox often used to design Kohonen's networks in classical hardware architectures. Therefore, we have adopted a communication technique based on the use of *Networks on Chip (NoC)*. This technique avoids the limitations of flexibility and extensibility often caused by using the point-to-point communications. On the other hand, the processing of the Kohonen algorithm has been fully distributed to avoid the architectural dependence of the overall *SOM* processing.

Based on this approach, we proposed an adaptable and extensible hardware *SOM* architecture dedicated to *FPGA* implementation. By a simple configuration, this architecture allows to reconfigure the SOM structure (the number of neurons, their distribution and the size of the weight vector) according to the needs of a specific application, without any design efforts. Moreover, we proposed a novel hardware architecture of a growing SOM during the learning phase.

Keywords: Artificial neural network (ANN), Self-Organizing Maps (SOM), Network-on-Chip (NoC), multiprocessor system-on-chip (MPSoC), Self-adaptive architecture, Evolving Artificial Neural Networks

Sommaire

Table des figures

Liste des tableaux

Introduction générale

Sommaire

1 Contexte général

L'évolution dans le domaine de la micro-électronique a connu une accélération exponentielle ces dernières décennies. Les progrès technologiques ont permis d'intégrer des systèmes de plus en plus complexes sur une seule puce, en leur offrant les performances nécessaires pour la manipulation rapide d'une grande quantité de données complexes et hétérogènes. Avec cette évolution, les systèmes électroniques et informatiques se sont imposés dans notre vie quotidienne dans différents domaines, pour mieux assurer le confort des utilisateurs à travers des différentes utilités qu'ils proposent (communication, analyse, supervision, etc...). De nos jours, on trouve des systèmes embarqués, placés sur des circuits intégrés d'échelles centimétriques, permettant d'assurer des fonctions de plus en plus complexes, diverses et exigeantes en termes de temps de réponse, de consommation d'énergie, de surface des supports, d'adaptabilité, etc... Afin de répondre à toutes ces contraintes, les concepteurs de circuits se retrouvent dans l'obligation de proposer des nouvelles approches de conception permettant non seulement d'effectuer les traitements de données au sein de ces systèmes sur puce mais également d'assurer d'autres fonctionnalités telles que l'acquisition des données, leurs pré-traitements, les tâches de communication, etc...

Il existe plusieurs approches de modélisation des systèmes d'information, parmi lesquelles deux approches fréquemment utilisées se distinguent : l'approche algorithmique et l'approche basée sur la connaissance [3,4]. La première approche est privilégiée lorsque le processus de traitement à effecteur est connu au préalable. Ainsi, elle demande une clairvoyance du concepteur pour prévoir tous les cas possibles du traitement envisagé. Par conséquent, pendant la phase de conception, le problème doit être maitrisé et la solution doit être prévue à l'avance. Par ailleurs, dans le cas des problèmes complexes, cette approche de modélisation est couteuse et même impossible dans certains cas. D'autre part, la seconde approche basée sur la connaissance permet de traiter les données suivant un certain nombre de règles logiques imposées au préalable par des experts humains dans le domaine. C'est un moyen d'employer et de stocker la connaissance

des experts sous une forme explicite [5]. Ces approches sont connues, dans le domaine de l'intelligence artificielle, sous le nom des *systèmes experts*. Elles sont souvent utilisées comme des modèles de prédiction et d'aide à la décision [6].

L'utilisation des deux approches de modélisation ne permet pas de garantir un succès promis, notamment dans la modélisation des problèmes complexes, difficiles à modéliser et souvent caractérisés par de nombreuses situations imprévues. Dans ce cas de figure, les approches de modélisation traditionnellement utilisées ne sont plus adaptées et doivent être remplacées ou complétées par d'autres paradigmes.

Dans la nature, on remarque que les êtres vivants ont une capacité de traiter des problèmes complexes d'une façon aisée paraissant facile pour les observateurs. Les biologistes ont essayé de comprendre le mécanisme étant à l'origine de ce traitement d'informations chez certains êtres vivants. Ces recherches ont permis de dévoiler l'existence d'un mécanisme neuro-biologique sophistiqué chez les mammifères, localisé au niveau du cerveau et appelé réseau de neurones. Ce mécanisme a montré une bonne performance de traitement d'information hétérogène ainsi qu'un bon niveau de tolérance aux erreurs éventuelles [7]. Ceci a inspiré les technologues pour proposer des systèmes numériques de traitement d'information émulant ce mécanisme biologique. On parle dans ce cas des réseaux de neurones artificiels. Cette approche, classée comme une des meilleures solutions de modélisation et par conséquent de résolution de problèmes complexes de l'intelligence artificielle, permet de résoudre des problèmes de reconnaissance, de classification, d'aide à la décision, d'extraction et réparation des données, etc... [8]

2 Motivation

Les modèles neuronaux bioinspirés sont composés d'un ensemble d'éléments de traitement inter-connectés communiquant entre eux pour réaliser le traitement prévu, en passant tout d'abord par une phase d'apprentissage suivie d'une phase de rappel. On trouve plusieurs modèles de réseaux de neurones artificiels dont les implémentations existent sous forme logicielle, matérielle et/ou mixte. L'avantage d'une implémentation logicielle est exprimé en termes de sa flexibilité et de sa capacité d'adaptation au traitement souhaité ainsi qu'à l'environnement externe pouvant être dynamique et variable. D'un autre côté, la limitation principale d'une implémentation logicielle d'un réseau de neurones artificiel réside dans le temps d'exécution pouvant être long et ainsi inacceptable pour une gamme d'application [9].

En général, pour des applications temps réel dont les contraintes temporelles sont serrées, le choix se porte souvent sur des solutions matérielles permettant d'une part d'exploiter le taux de parallélisme intrinsèque des réseaux de neurones artificiels et d'autre part d'améliorer les performances d'exécution nécessaires pour répondre à ces contraintes de temps très fortes [10]. Par contre, les implémentations matérielles sont limitées par le manque de flexibilité des architectures implantées. De ce fait, la plupart des architectures matérielles des réseaux de neurones proposées dans la littérature sont des architectures dédiées à des applications spécifiques. Selon les travaux

rapportés dans la littérature, il est difficile voire impossible d'adopter la même architecture matérielle d'un réseau de neurone à une variété de fonctions ayant des différents besoins de traitement. Ceci est essentiellement dû à la diversité des paramètres d'un réseau de neurones artificiels : le nombre de neurones sur le réseau, leur topologie d'interconnexion, la nature des données à l'entrée, temps de traitement, taille de la mémoire, etc…, qui sont généralement fixés au cours de la conception de l'architecture matérielle du réseau de neurones artificiels en question. Par conséquent, le changement de l'application nécessite un effort considérable de conception pour adapter les paramètres de l'architecture du réseau de neurones artificiels aux besoins de la nouvelle application.

Ce manque de flexibilité est dû d'une part, à l'utilisation des techniques de communications classiques pour l'échange des informations entre les unités de calcul de l'architecture représentant les neurones du réseau et d'autre part, à la dépendance architecturale de certaines unités de traitement de l'architecture dont le comportement est spécifique aux paramètres de l'application traitée. Éventuellement, pour certaines applications temps réel comme par exemple dans le cas des applications de traitement d'images et d'analyse des signaux physiologiques, l'aspect hétérogène des données de l'environnement externe demande une adaptation dynamique du réseau de neurones artificiels ceci étant difficile et même impossible avec les architectures matérielles classiques reportées dans la littérature. Ces limites sont une des causes principales du refus d'adoption massive de ce modèle de calcul dans certaines applications des systèmes embarqués dont les besoins architecturaux, généralement définis à l'avance dans la phase de conception, peuvent évoluer au cours du fonctionnement du système [11].

Dans les travaux de recherche menés dans cette thèse, uniquement le modèle neuronal de type carte auto-organisatrice a été considéré. Dans ce cadre et dans l'optique des limitations des architectures matérielles des réseaux de neurones artificiels évoquées précédemment, vient l'idée de proposer une architecture matérielle flexible, adaptable et extensible du modèle neuronal « *carte auto-organisatrice de Kohonen* » en anglais « *Self-Organising Map* (*SOM*) ». Dans ce contexte, se situent nos travaux de recherche qui s'insèrent dans le cadre d'une collaboration franco-tunisienne entre le laboratoire de Technologie et imagerie médicale (*LR12ES06*) du côté tunisien, avec une expertise dans l'utilisation des architectures matérielles des réseaux de neurones artificiels pour des applications médicales, et l'institut Jean Lamour (*UMR 7198*) du côté français, avec une expertise dans les implémentations des architectures reconfigurables et auto-organisées. Le but final de ces travaux est de proposer un système auto-adaptatif embarqué permettant le traitement et la classification des données hétérogènes.

3 Contribution

Les contributions principales des travaux de recherche menés au cours de cette thèse, dont l'objectif principal est d'ajouter un certain niveau de flexibilité et d'adaptabilité aux architectures matérielles des cartes *SOM* tout en respectant les performances temporelles offertes par leur

implémentation matérielle, sont les suivantes :

— Une nouvelle approche de conception d'architecture matérielle des cartes *SOM* : Dans cette approche proposée, les communications point à point souvent utilisées par les architectures classiques, étant également une cause principale de manque de flexibilité de ces dernières, ont été remplacées par la technique de communication à base des réseaux sur puce « Network on chip (*NoC*) ». Cette approche proposée nommée *SOM-NoC* permet d'ajouter un certain niveau de flexibilité et d'adaptabilité d'une part, et d'offrir la possibilité d'extensibilité aux architectures matérielles d'autre part [12]. L'architecture *SOM-NoC* introduit une adaptabilité non seulement au niveau neurone mais également au niveau de la structure globale. Ainsi, en plus de l'auto-organisation assurée par l'algorithme du réseau de neurones artificiels réalisé par l'architecture *SOM-NoC*, elle permet de s'auto-restructurer par un simple paramétrage de manière dynamique à la volée (*online*) en fonction des échantillons de l'environnement externe reçus à l'entrée du système.

— Une nouvelle architecture nommée *SWAP-SOM* basée sur l'approche *SOM-NoC* permettant de traiter en même temps plusieurs applications différentes nécessitant les cartes auto-organisatrices : L'architecture proposée permet d'identifier à la volée de manière dynamique les paramètres spécifiques à l'application donnée et d'adapter le comportement de tous ses modules (neurones) à l'exécution de cette application. La reconfiguration de l'architecture d'une application vers une autre se fait en un seul cycle d'horloge à la réception et l'analyse des données d'entrée. L'architecture SWAP-SOM a été validée sur un système de compression d'image effectuant en parallèle une exploration des paramètres de compression à utiliser pour une classe d'image donnée.

— Une nouvelle architecture matérielle dynamique et évolutive de la carte *SOM* nommée *GG-SOM* : L'architecture *GG-SOM*, également basée sur l'approche *SOM-NoC*, est une architecture implémentant la carte *SOM* à structure variable dans le temps capable de s'auto structurer au cours de la phase d'apprentissage, afin d'aboutir à une structure globale adaptée représentant de manière optimale l'environnement externe d'une application donnée. Autrement dit, la taille de la carte *SOM* à utiliser par l'architecture *GG-SOM* dépendra uniquement des spécificités de l'application traitée et ressources matérielles disponibles pour son implantation et sera déterminée de manière dynamique au cours du fonctionnement. Cette architecture originale est à notre connaissance la première architecture matérielle d'une carte auto-organisatrice *SOM* évolutive proposée dans la littérature.

4 Plan du manuscrit

Pour présenter les travaux de recherche et détailler les approches proposées au cours de cette thèse, ce manuscrit est structuré de la manière suivante :

Le premier chapitre est consacré à la présentation des réseaux de neurones artificiels. Dans ce chapitre, tout d'abord le principe fondamental des outils connexionnistes est introduit. Cette

introduction est suivie par la présentation des modèles neuronaux fréquemment utilisés, en expliquant la stratégie de choix du modèle neuronal convenable selon les besoins fonctionnels des applications. Enfin, une présentation détaillée de la carte auto-organisatrice de Kohonen conclut ce chapitre.

Le deuxième chapitre est entièrement dédié aux approches d'implémentation de la carte auto-organisatrice. Dans ce chapitre bibliographique, les différentes approches d'implémentations, logicielles et matérielles, rapportées dans la littérature sont présentées. Le chapitre commence par les approches d'implémentation logicielle, en présentant leurs avantages et leurs inconvénients. De plus, des approches logicielles avancées permettant de surmonter les problèmes de performances des implémentations logicielles dites classiques sont également présentées dans cette partie. Ensuite, une étude détaillée des différentes approches d'implémentation matérielle des cartes *SOM* est présentée. Cette partie est conclue par la présentation des inconvénients majeurs des approches matérielles des cartes *SOM* étant leur manque de flexibilité et d'extensibilité, en citant également quelques travaux mentionnés dans la littérature proposant des solutions à ces inconvénients.

Le troisième chapitre présente les choix architecturaux effectués pour aboutir à une architecture matérielle de la carte *SOM* qui est à la fois adaptable, flexible et extensible. La technique de communication à base des réseaux sur puce adoptée pour la conception de l'architecture matérielle proposée dans ces travaux de recherche est présentée au début de ce chapitre. Ensuite, l'architecture matérielle nommée *SOM-NoC* permettant d'améliorer la flexibilité et l'extensibilité des architectures matérielles de la carte auto-organisatrice est détaillée. Enfin, les résultats de validation de cette architecture sur l'exemple d'une application de compression d'image sont présentés.

Le quatrième chapitre est consacré à la présentation des architectures matérielles évolutives de la carte *SOM* proposées dans ces travaux de recherche. Les deux architectures matérielles proposées dans ce chapitre utilisent comme point de départ l'architecture *SOM-NoC* détaillée dans le troisième chapitre. La première partie de ce chapitre présente l'architecture matérielle nommée *SWAP-SOM* capable d'exécuter plusieurs applications simultanément sur le même support physique. La deuxième partie est consacrée à la présentation de l'architecture matérielle nommée *GG-SOM*, étant une architecture innovante de la carte auto-organisatrice à structure croissante au cours de l'apprentissage. Il s'agit à notre connaissance de la première architecture matérielle évolutive de la carte *SOM* proposée dans la littérature. Ce chapitre se termine avec la présentation des résultats de validation obtenus à base de ces deux architectures.

La dernière partie consiste en une **conclusion générale et perspectives**, dans lesquelles le bilan des travaux réalisés au cours de cette thèse en soulignant les contributions majeures proposées et en dégageant les perspectives ouvertes pour des éventuels travaux futurs sont présentés.

Chapitre 1

Généralités sur les réseaux de neurones artificiels et les réseaux de Kohonen

Sommaire

"*Les prouesses réalisées par des individus exceptionnels, grâce à leur art et à leur intelligence, tôt ou tard la technologie les rend possibles à tout le monde.*"

Roland Topor

1.1 Introduction

Le comportement du cerveau chez les mammifères a toujours ébloui les scientifiques par sa façon de traiter une diversité d'informations et sa tolérance aux erreurs. Les recherches effectuées dans le domaine physiologique ont promis de lever progressivement le voile sur ce mystère biologique et d'exposer le mécanisme principal neuro-physiologique en expliquant son comportement. Cette découverte biologique a inspiré les technologues de modéliser et concevoir de systèmes artificiels à l'image de ces systèmes biologiques. Ces systèmes sont capables de résoudre des problèmes dont les approches algorithmiques sont trop complexes voire impossibles à résoudre compte tenu du nombre important de paramètres les caractérisant.

Plusieurs solutions d'émulation ont été proposées dans la littérature. De nos jours, il existe plusieurs axes dans ce domaine de recherche, parmi lesquels, on trouve le *neuro-mimétisme* et le *connexionnisme*. Dans le premier axe, les neuro-miméticiens modélisent les réseaux de neurones artificiels, tout en restant fidèles aux modèles biologiques du système nerveux. Ils essaient de reproduire le fonctionnement du réseau de neurones biologique afin de mimer le raisonnement humain [13]. Les modèles neuronaux bio-inspirés émulent le mode de circulation des informations du système nerveux central [14]. Ces approches sont fréquemment utilisées pour des applications de la navigation robotique [15, 16], et des applications bio-médicales [17].

D'un autre coté, les travaux de recherche relatifs au deuxième axe ne cherchent pas à imiter strictement le modèle biologique, mais s'inspirent seulement de la structure biologique des réseaux de neurones [18–20]. Dans ce cadre, on trouve des approches dites connexionnistes cherchant à proposer des systèmes inspirés de la définition des réseaux de neurones biologiques. L'idée finale est donc, d'améliorer la capacité des systèmes actuels en y ajoutant des modules de traitement fortement inter-connectés contribuant au développement de cette couche bio-inspirée et permettant d'assurer en parallèle un traitement global difficilement réalisable à base des approches d'analyse d'information classiques [21]. Les réseaux de neurones artificiels proposés dans l'axe de connexionnisme sont généralement utilisés pour des applications de classification et de prévision [19, 21].

Les travaux présentés ici se classent dans le domaine des réseaux de neurones artificiels connexionnistes. Ce chapitre est essentiellement consacré à la présentation de ce modèle bio-inspiré. Après quelques définitions et un petit historique sur ce domaine de recherche, le chapitre

présentera le neurone formel, son apprentissage et son fonctionnement ainsi que les différents modèles des réseaux de neurones artificiels. La dernière partie, avant de conclure sera consacrée aux réseaux de Kohonen essentiellement à la carte auto-organisatrice, objet principal des travaux de recherche menés dans le cadre cette thèse.

1.2 Définitions

Les lecteurs peuvent se référer à plusieurs ouvrages définissant et expliquant le principe des réseaux de neurones artificiels. Une introduction au réseau de neurones formels (ou artificiels) est présenté par *Kriesel* dans son livre [7], avec une description détaillée des outils neuronaux et de leurs fonctionnements. Une présentation plus détaillée des réseaux de neurones artificiels est proposée par *Krose et Van Der Smagt*. Dans leur livre [21], une introduction fondamentale et théorique a été proposée. D'autre part, ils ont présenté différents domaines d'application des réseaux de neurones ainsi qu'une brève présentation de leurs implémentations matérielles et logicielles.

Avant de présenter le concept technique des réseaux de neurones artificiels tel qu'il est proposé dans la littérature, commençons par la définition des mots « réseau », « neurone » et « artificiel » dans la langue de Molière :

Vocabulaire 1 (Nom) : Un *réseau* est un ensemble organisé dont les éléments, répartis en divers points, sont inter-connectés afin de garantir leur communication [22].

Vocabulaire 2 (Nom) : Un *neurone* est une cellule à base de tissu nerveux, capable de recevoir, d'analyser et de produire des informations [22].

Définition 1 : Un *réseau de neurones* est un graphe orienté, constitué d'un ensemble d'unités, réalisant des traitements élémentaires, structurées sous forme de couches successives interconnectées capables d'échanger des informations via des liens structurés [23].

Les réseaux de neurones artificiels inspirés du concept biologique n'étaient qu'une modélisation mathématique de ce graphe. Ils ont donc reproduit certaines caractéristiques d'analyse neurobiologique (la capacité d'apprentissage, de mémoriser l'information et de traiter des informations incomplètes ou corrompues). Plusieurs modèles neuronaux ont été proposés depuis l'introduction de cette notion. Les modèles artificiels proposés de point de vue connexionniste se basent sur la structure et le comportement global du modèle biologique. D'où, le principe d'interconnexion et de parallélisme est toujours respecté sans tenir compte du principe de la structuration. A base de ce principe, les réseaux de neurones ont connu leur succès [18].

Vocabulaire 3 (Adjectif) : Un produit *artificiel* est le résultat d'une manipulation [22].

Définition 2 : Les *réseaux de neurones artificiels* sont des réseaux fortement connectés de processeurs élémentaires fonctionnant en parallèle. Chaque processeur élémentaire calcule une sortie unique sur la base des informations qu'il reçoit en entrée. Toute structure hiérarchique de réseaux est évidemment un autre réseau.

1.3 Historique

Avec l'apparition du principe de la cybernétique, l'objectif principal était de donner à la machine un certain niveau d'intelligence semblable à celle de l'être humain. L'histoire des réseaux de neurones artificiels a commencé il y a *75 ans*. Le tableau 1 présente les dates significatives dans l'histoire de l'évolution des réseaux de neurones.

Avec des hauts et des bas, le vrai succès n'a vu le jour qu'il y a une trentaine d'années [7]. Depuis, les réseaux de neurones artificiels se sont imposés comme l'une des meilleures approches d'analyse d'informations hétérogènes. Elle se présente aussi comme une solution aux problèmes dont la formalisation est complexe voire impossible. De ces faits, les réseaux de neurones artificiels sont utilisés fréquemment dans le domaine de la prévision, de la reconnaissance des formes et de la reconstruction et correction des données ainsi que dans le domaine de la robotique.

TABLEAU *1 – Dates significatives dans l'histoire d'évolutions des réseaux de neurones artificiels*

Date	***Auteurs***	***Evénement***
1890	*James*	Présentation du concept de la mémoire associative - Loi de fonctionnement pour l'apprentissage.
1943	*McCulloch et Pitts*	Modélisation du neurone biologique en neurone formel.
1949	*Hebb*	Proposition de la *règle de Hebb*.
1957	*Rosenblatt*	Proposition du modèle du *Perceptron* - Premier neuro-composant.
1960	*Widrow*	Développement du modèle ADALINE inspiré du Perceptron.
1969	*Minsky et Papert*	Mise en avant des limites du Perceptron - Abandon des recherches.
1967-1981	*Grossberg, Kohonen, etc...*	Période de l'ombre - Poursuite déguisée des recherches.
1982	*Hopfield*	Relance des recherches avec la présentation du *modèle de Hopfield* - Théorie du fonctionnement et des possibilités des réseaux de neurones.
1983	*Fukushima, Miyake et Ito*	Proposition du modèle neuronal *Neocognitron* utilisé pour la reconnaissance des caractères manuscrits.
1985-1986	*Parker, Rumelhart et LeCun*	Développement des réseaux de neurones multicouches et proposition de l'algorithme de la *rétro-propagation du gradient*.

En *1943*, *McCulloh et Pitts* [24] ont proposé le premier modèle d'un neurone formel : un neurone au comportement binaire. Leur modèle a été inspiré du modèle de neurone biologique présenté par *James en 1890*. Ce dernier avait introduit le concept de la mémoire associative. Ensuite, avec le modèle formel, *McCulloh et Pitts* ont démontré la possibilité de résoudre des problèmes logiques arithmétiques complexes à l'aide du concept des réseaux de neurone. L'idée suppose que le cerveau est équivalent à une machine de *Turing*.

En *1949*, le célèbre physiologiste américain *Hebb* [25] a repris l'idée de *James*. A base de la théorie de *Pavlov*, il a présenté une règle d'apprentissage des réseaux de neurones. On parle de *la*

règle de Hebb utilisée par plusieurs algorithmes neuronaux. L'idée s'appuie sur la modification des caractéristiques des connexions inter-neuronales (souvent appelées poids) au cours de la phase d'apprentissage.

Les premiers succès des réseaux de neurones artificiels apparurent en *1958*, avec le célèbre modèle du *Perceptron* proposé par *Rosenblatt* [26]. En s'inspirant du système visuel, il a réussi à développer un réseau de neurones artificiels à deux couches : une couche de perception (*rétine*) et une couche de la prise de décision. Vu les moyens technologiques de l'époque, c'était un exploit de faire fonctionner un système artificiel capable d'apprendre par l'expérience. Un deuxième exploit a vu le jour deux ans plus tard. Un automaticien nommé *Widrow* présente le modèle de l'*ADAptative LINear Element* (*ADALINE*) [27]. Avec une structure semblable à celle du perceptron, *ADALINE* intègre une nouvelle fonction d'apprentissage connue de nos jours sous le nom de l'algorithme de *rétro-propagation du gradient*.

Après un succès relatif, la publication de *Minsky et Papert* [28] en *1969* entraine un désintérêt pour ce domaine. L'étude qu'ils publient met en exergue les limites des Perceptrons à traiter des problèmes non linéaires. Cette critique a eu une grande influence sur le détournement des recherches (surtout sur le plan financier) principalement vers d'autres approches telles que les systèmes à base des règles. Pendant *13 ans* les recherches dans le domaine des réseaux de neurones artificiels sont restées dans l'ombre. Mais certains chercheurs tels que *Kohonen et Grossberg*, poursuivaient leurs recherches déguisées sous le couvert de diverses appellations (traitement adaptatif du signal, reconnaissance de formes, modélisation en neurobiologie, etc...).

Le renouveau de la recherche sur les réseaux de neurones artificiels est arrivé en *1982*. Le physicien *Hopfield* [29] a présenté des études proposant une nouvelle théorie de fonctionnement permettant d'augmenter les possibilités de traitement avec les réseaux de neurones artificiels. Ce nouveau modèle à base d'apprentissage bouclé (encore utilisé aujourd'hui pour certains algorithmes neuronaux) a relancé l'intérêt sur les réseaux de neurones artificiels sans avoir pour autant levé le problème des limites exposées par *Minsky et Papert*. Dans la même année, *Kohonen* [30] propose sa carte auto organisatrice « *SOM* » comme l'une des meilleures approches de classification. D'autre part, en *1983* *Fukushima et al.* introduisent un nouveau modèle neuronal appelé *Neocognitron* utilisé pour la reconnaissance des caractères manuscrits [31].

La levée des limites du perceptron est obtenue avec la proposition de la machine de *Boltzmann en 1983*. En revanche, l'utilisation pratique de ce modèle est rendue difficile par des temps de convergence extrêmement longs. Entre *1985 et 1986*, l'avancée significative apparait avec la proposition de l'algorithme de rétro-propagation du gradient. Cette théorie a été développée de manière indépendante par trois groupes de chercheurs : *Parker* [32], *Rumelhart* [33] et Lecun [34]. L'algorithme proposé est aujourd'hui la base de la fonction d'apprentissage du célèbre *Perceptron Multi-Couches (MLP)* fréquemment utilisé depuis cette découverte. Et depuis les recherches ont été relancées de façon intensive.

Les réseaux de neurones connaîtront un grand essor au cours des *années 90*. Plusieurs modèles basés sur l'algorithme du perceptron et de celui de la rétro-propagation du gradient sont proposés.

Des recherches s'intéressent aussi à la notion de réseaux de neurones évolutifs et dynamiques [35].

1.4 Concept neuro-biologique

La capacité de traitement d'informations chez les mammifères se base sur l'existence d'un réseau complexe de cellules nerveuses interconnectées permettant de coordonner tout un système moteur [35]. En outre, une action musculaire commence par l'interception des données physiques de l'environnement qui seront converties sous forme d'influx nerveux circulant dans le réseau sur un circuit sélectif. Ce circuit est composé d'un ensemble d'unités appelées neurones [36].

Définition 3 : Un *neurone biologique* est une cellule excitable constituant l'unité fonctionnelle de base du système nerveux. Les neurones assurent la transmission d'un signal bio-électrique appelé *influx nerveux*.

Dans le système nerveux des êtres vivants, on peut compter entre quelques milliers à plusieurs milliards de neurones distribués sur tous le corps. La structure d'un neurone, telle qu'elle est schématisée sur la figure 1, est composée d'un corps cellulaire avec un noyau dont le rôle est d'activer ou non le neurone selon l'excitation reçue. D'un côté, le corps cellulaire se ramifie pour constituer des arborescences dendritiques. Sur ces arborescences, on peut compter des milliers de dendrites qui serviront pour l'acheminement des informations en entrée vers le corps cellulaire nommé aussi Soma au niveau duquel tous les signaux d'entrée seront accumulés. D'un autre côté, le corps cellulaire se prolonge pour former la sortie du neurone nommée axone. Cette dernière pouvant avoir une longueur de quelques centimètres, se termine par une arborescence de synapses qui serviront comme points de contact avec les neurones adjacents.

La circulation d'influx nerveux passe par des phénomènes électriques et d'autres chimiques avec une procédure de transmission complexe. Lorsqu'un neurone est activé, une impulsion électrique appelée potentiel d'action (*Spike*) sera émise tout au long de son axone. Au niveau de l'arborescence synaptique, les synapses actionnent des canaux ioniques pour libérer des ions neurotransmetteurs. Ces derniers viennent se fixer sur les récepteurs au niveau des dendrites du neurone voisin. Dans le cas d'une synapse excitatrice, des canaux ioniques s'ouvrent pour laisser circuler des ions et donc produire un signal électrique qui sera acheminé vers le *Soma* via les dendrites. Par contre, ces canaux ioniques se ferment, si c'est le cas d'une synapse inhibitrice, pour réduire la circulation des ions. Par conséquent, plus les canaux ioniques sont ouverts plus l'intensité du signal électrique est élevée. C'est le phénomène de pondération des influx nerveux, au niveau des dendrites, par les synapses. Enfin, au niveau du soma, les signaux en entrée seront accumulés jusqu'à atteindre un seuil permettant d'activer le neurone. La réponse neuronale est de type binaire non-linéaire qui dépend seulement de l'intensité des signaux électriques pondérés. L'apprentissage est donc basé sur l'organisation des liaisons synaptiques.

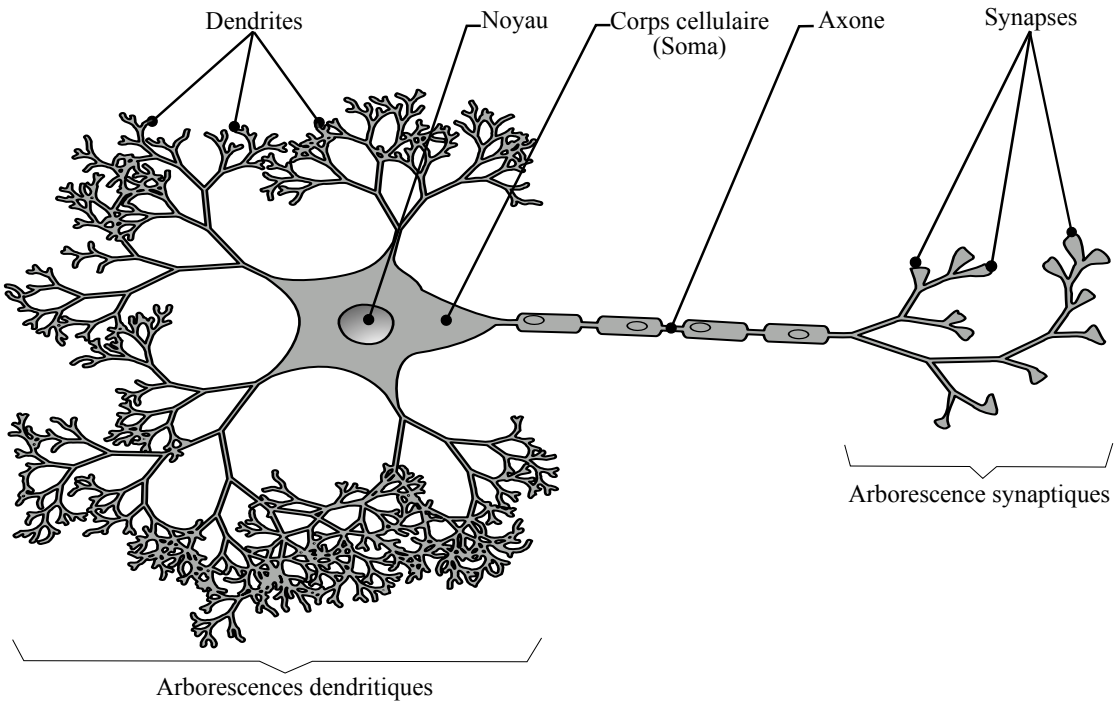


FIGURE *1 – Structure d'un neurone biologique*

1.5 Neurone formel

Vocabulaire 4 (Adjectif) : Un système *formel* est posé explicitement dans une théorie déductive de règles de formation des propositions et de règles de déduction suivant lesquelles on raisonne [22].

Vocabulaire 5 (verbe) : *Formaliser* est réduire une théorie à un système formel [22].

Définition 4 : Un *neurone formel* est une forme mathématique représentée par des outils numériques pour émuler le comportement d'un neurone biologique.

1.5.1 Structure

Un neurone formel est présenté en tant qu'unité fonctionnelle élémentaire. L'architecture d'un neurone formel telle qu'elle est présentée sur la figure 2, est inspirée de son homologue biologique. Un neurone formel est alimenté par un vecteur d'entrée $\overrightarrow{X}$ de dimension *D*, $\overrightarrow{X} = (x_1, x_2, ..., x_D)$ dont un poids de connexion w_i (*weigth*) est associé à chaque attribut du vecteur. Les *attributs en entrée* ainsi que leurs *poids de connexion* d'un neurone formel correspondent respectivement aux dendrites et synapses sur son homologue biologique [23].

Définition 5 : Le *poids de connexion* est une valeur $w_{i,j}$ associée à l'attribut d'entrée *j* d'un neurone *i*. Il correspond au poids synaptique du neurone biologique.

Définition 6 : La *fonction de combinaison* d'un neurone est une fonction d'accumulation des différentes entrée de ce neurone émettant comme résultat leur somme pondérée. Cette valeur

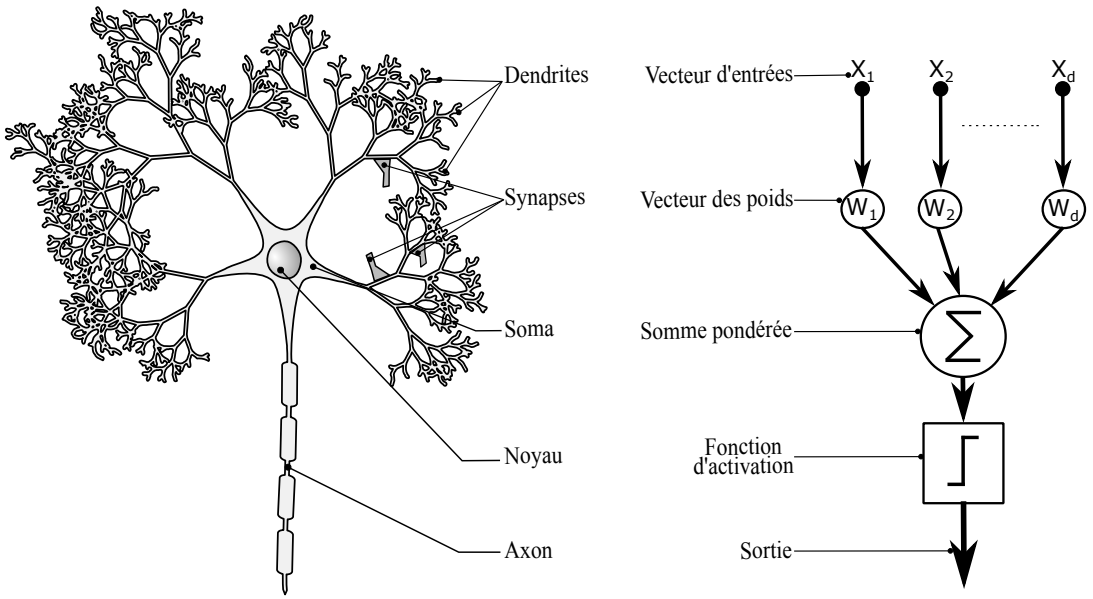


FIGURE 2 – *Mise en correspondance neurone biologique / neurone artificiel*

servant comme paramètre d'activation du neurone est appelée aussi *potentiel membranaire*.

Enfin, la somme calculée sera envoyée à une fonction d'activation pour générer la valeur de la sortie du neurone. La fonction d'activation reliée à la sortie traduit le noyau et l'axone sur la structure biologique d'un neurone.

Définition 7 : La *fonction de transfert* d'un neurone est la fonction qui calcule l'état (activé ou non) de ce neurone à partir du potentiel membranaire calculé par la fonction de combinaison. Cette fonction est appelée aussi *fonction d'activation*.

Toutes les valeurs en entrée sont récupérées au niveau d'un accumulateur permettant de réaliser une fonction de sommation ou de combinaison pour générer une somme pondérée des valeurs en entrée. Cette fonction de combinaison correspond évidement au *corps cellulaire (Soma)* dans la structure d'un neurone biologique.

1.5.2 Comportement

Pendant le fonctionnement d'un réseau de neurones, un neurone formel produit une sortie variable en fonction des attributs en entrée provenant de l'environnement externe ou des neurones voisins selon son emplacement dans le réseau. La variable en sortie est une valeur *différenciée* permettant de distinguer l'état actif ou l'état inactif du neurone [35].

A la réception des attributs en entrée, ces derniers passent par une fonction de pondération avec les poids des connexions. Ensuite, toutes les valeurs pondérées seront transmises à la fonction de combinaison. Ainsi la somme pondérée est calculée.

En fonction de la pondération, on distingue deux types d'unités neuronales [37, 38] :

Unité produit scalaire : où la pondération entre le vecteur d'entrée et le vecteur des poids de

connexions est un produit scalaire. La somme pondérée est calculée par l'équation suivante :

$$P_s = \sum_{i=1}^{D} (w_i \cdot x_i) \tag{1}$$

Unité distance : où la pondération calculée représente la distance entre le vecteur des poids de connexion et le vecteur d'entrée. Cette distance peut être calculée sur différents niveaux (L1 : Manhattan, L2 : Euclidienne, L3 ...). L'équation 2 présente la formule de calcul de la distance Euclidienne :

$$D_{L2} = \sum_{i=1}^{D} (w_i - x_i)^2 \tag{2}$$

Enfin, la somme sera envoyée à la fonction d'activation reconnue souvent par la fonction de transfert [8]. Cette fonction permet de générer la valeur de la sortie en fonction d'un seuil introduit comme paramètre. Dans la littérature on trouve plusieurs formes de la fonction de transfert. Une étude détaillée des fonctions d'activation est proposée dans [39]. Le tableau 2 présente les fonctions d'activation les plus utilisées.

Pour résumer, un neurone calcule sa valeur en sortie grâce à l'équation 3. La forme de cette équation traduit la non-linéarité du traitement à base de réseaux de neurones. La valeur en sortie présente l'état du neurone qui sera émis aux neurones en aval ou directement à l'environnement extérieur.

$$S = f\left(\sum_{i=1}^{D} \|x_i \cdot w_i\|\right) \tag{3}$$

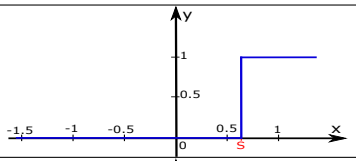
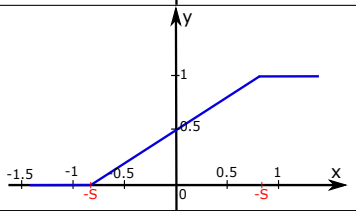
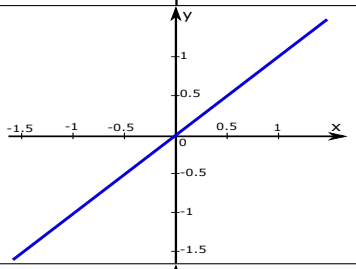
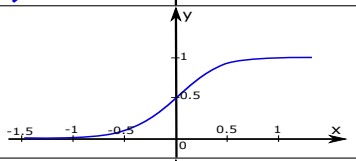
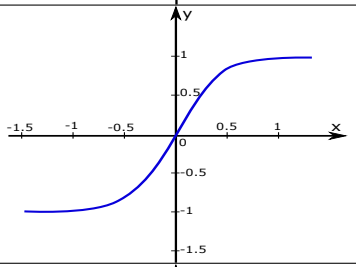
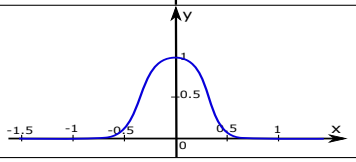
1.6 Apprentissage

L'apprentissage est la fonction la plus intéressante des réseaux de neurones. C'est la base d'inspiration de cette approche. Un réseau de neurones est capable d'adapter son fonctionnement, dans un environnement défini, à base d'un algorithme d'apprentissage. Qu'est-ce que l'apprentissage ? Il est difficile de répondre à cette question, car on ne trouve pas une définition universellement adoptée. Les multiples définitions de la littérature expliquent différentes notions situées autour du même concept.

Vocabulaire 4 (Nom) : L'*apprentissage* est un ensemble de processus de mémorisation mis en œuvre par l'animal ou l'homme pour élaborer ou modifier les schémas comportementaux spécifiques sous l'influence de son environnement externe et de son expérience [22].

Définition 8 : L'*apprentissage* est une phase de transition entre un état d'ignorance et un état de savoir afin d'adapter le comportement du système de gestion des données à l'environnement externe [32].

TABLEAU 2 – *Différentes formes de la Fonction d'activation*

Fonction	***Comportement***	***Formule mathématique***
Pas unitaire		$f(x) = \begin{cases} 0, & si\ x < S \\ 1, & si\ x \geq S \end{cases}$
Linéaire Seuillée		$f(x) = \begin{cases} 0, & si\ x \leq -S \\ m \cdot x + b, & si\ -S < x < S \\ 1, & si\ x \geq S \end{cases}$
Identité		$f(x) = x$
Sigmoïde		$f(x) = \frac{1}{e^{-x}+1}$
Tangente		$f(x) = \frac{e^{x}-e^{-x}}{e^{x}+e^{-x}}$
Gaussienne		$f(x) = \frac{1}{\sqrt{2\tilde{}\sigma}} * e^{\frac{-(x-m)^2}{2\sigma^2}}$

Définition 9 : L'*apprentissage* d'un réseau de neurones consiste à créer un circuit logique entre un stimulus et la réponse correspondante [40].

Définition 10 : L'*apprentissage* est un processus dynamique et itératif permettant de modifier les paramètres d'un réseau en réaction avec les stimuli qu'il reçoit de son environnement externe. Le type d'apprentissage est déterminé par la manière dont les changements de paramètre surviennent [41].

En général, un réseau de neurones doit être capable de réaliser une fonction particulière dans un environnement bien déterminé. Cette capacité est acquise grâce à l'algorithme d'apprentissage adopté par le modèle neuronal. C'est la première phase dans le cycle de vie d'un réseau de neurones (passage d'un état d'ignorance à un état de savoir). En cas de changement de fonction

ou d'environnement externe, il faut repasser par cette phase pour adapter le comportement du réseau de neurones aux nouveaux paramètres (transition entre deux états de savoir).

L'apprentissage d'un réseau de neurones consiste à faire subir au système un ensemble de stimulations itératives. Par intuition, le réseau sera informé de mieux en mieux sur l'ensemble d'entrée. A chaque itération d'apprentissage (alimentation du réseau de neurones par un stimulus), les neurones du réseau adaptent leurs traitements pour répondre aux besoins de la fonction ciblée.

A la base, l'idée d'apprentissage a été inspirée de l'apprentissage par conditionnement observé chez l'animal [42]. Le physiologiste *Pavlov* a introduit la notion de conditionnement classique en *1927* expliquée par sa fameuse expérience de conditionnement du chien illustré sur la figure 3 [13]. Le physiologiste russe remarquait que la présence de la nourriture est un stimulus inconditionnel provocant la salivation chez le chien. A partir de là, il a commencé à provoquer le son d'une cloche comme stimulus conditionnel, à chaque fois où il présentait la nourriture au chien. Après un certain nombre de répétitions de l'expérience, le son de la cloche devient associé à la nourriture. Ensuite, il a remarqué que le son de la cloche seul provoque la salivation. L'apprentissage consiste à conditionner un stimulus à une réponse en associant deux stimuli. A partir de cette expérience, *Hebb* a proposé sa règle d'apprentissage dont la technique consiste à renforcer les connexions entre les neurones actifs.

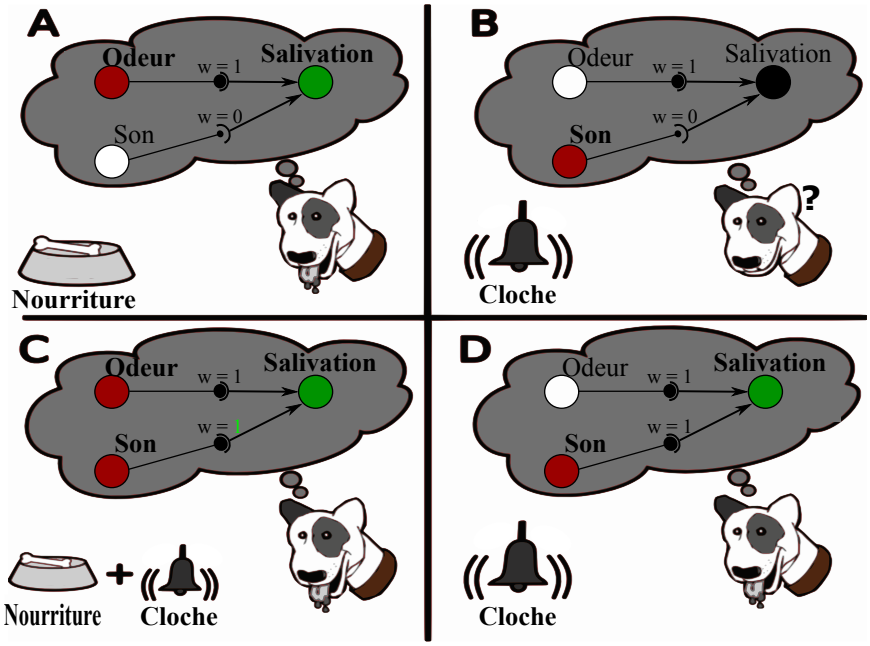


FIGURE 3 – *Expérience de Pavlov : Conditionnement classique : A - stimulus inconditionnel. B - stimulus neutre. C – Association de stimuli. D – Stimulus conditionnel.*

Dans la majorité des réseaux de neurones existants [43], le principe de fonctionnement est similaire voire identique au niveau de chaque neurone du réseau. Par conséquent, le comportement d'un neurone dépend des poids ainsi que du seuil d'activation qui lui est associé [44]. Ces valeurs sont appelées : *les paramètres libres* du modèle neuronal [45]. Ces paramètres définissent

le comportement d'un réseau de neurones sur un ensemble d'entrée. Il est difficile et même impossible de prédire les valeurs des paramètres libres pendant la phase de conception. D'où, le passage par la phase d'apprentissage est obligatoire. Pendant cette phase, les paramètres libres seront modifiés afin de réduire l'erreur de la réponse du système aux exemples présentés à son entrée et à l'expérience.

Après un certain nombre d'itérations d'apprentissage, la marge d'erreur diminue pour qu'elle soit tolérée par le système. A ce moment, le réseau de neurones n'a plus besoin d'apprentissage. Par conséquent on peut dire que le réseau de neurones est capable d'analyser des exemples réels de sont environnement externe en tant que expert. A ce niveau, la *phase de décision*, souvent appelée *phase du rappel*, commence. Il existe quelques modèles neuronaux où l'apprentissage est permanent. Dans ce cas, l'apprentissage continu pendant la phase de rappel, si le réseau reçoit un éventuel échantillon non reconnu. Par contre, pour la plupart des réseaux de neurones, on peut distinguer la phase de rappel et la phase d'apprentissage. Cette dernière, se termine après avoir le comportement du réseau de neurones le mieux adapté pour la fonction cible. Cette technique permet d'éviter le sur-apprentissage qui peut toucher le comportement du réseau de neurones. Donc, pendant la phase de décision, le système conserve le comportement adapté au cours de la phase d'apprentissage indépendamment des changements de l'environnement externe.

Un modèle neuronal est caractérisé par le traitement réalisé par ses neurones, son architecture de connexion et sa règle d'apprentissage. Cette dernière définit l'algorithme à suivre pour adapter les paramètres libres du réseau à la fonction désirée. On distingue deux catégories de règles d'apprentissage : apprentissage supervisé ou non-supervisé.

1.6.1 Règle d'apprentissage supervisé

L'apprentissage supervisé ou actif consiste à créer une relation (correspondance) entre un stimulus et la réponse associée. Ainsi, la base d'apprentissage est présentée sous forme d'un ensemble de couples : stimulus et réponse associée (x_i , e_i). D'autre part, les neurones situés sur la couche de sortie du réseau sont étiquetés de manière à ce que pour une entrée de valeur bien déterminée x_i, le neurone activé à la fin de la phase d'apprentissage soit celui qui porte l'étiquette e_i correspondante à la réponse associée attendue pour l'entrée x_i.

La technique consiste à évaluer l'erreur pour chaque stimulus et modifier les paramètres libres des neurones afin de minimiser l'erreur dans les prochaines itérations. En cas de non correspondance, il faut affaiblir les poids synaptiques des neurones actifs. Par contre, en cas de correspondance entre le stimulus et la réponse, la procédure consiste à renforcer les poids des neurones actifs. Donc, un modèle neuronal à règle d'apprentissage supervisé passe par une procédure de calibration des paramètres libres pendant la phase d'apprentissage. Ensuite, les circuits mémorisés seront généralisés sur les entrées aléatoires pendant la phase de décision.

Différents domaines d'application sont associés aux algorithmes à base de règle d'apprentissage supervisé (aide aux diagnostics, analyse prédictive, analyse financière, détection de fraudes, etc…). Parmi les algorithmes d'apprentissage supervisé les plus répandus, nous pouvons citer la

règle Delta [27] et les algorithmes de *retro-propagation du gradient* très utilisés par le modèle *Perceptron Multi Couches* (*PMC*) [45].

Malgré la complexité de ces algorithmes ainsi que le coût de construction des échantillons d'apprentissage, l'intérêt des algorithmes d'apprentissage supervisés est la possibilité de transférer l'expertise de l'utilisateur à la machine. Et donc, faciliter les tâches réalisées manuellement en évitant les erreurs humaines.

1.6.2 Règle d'apprentissage non-supervisé

L'apprentissage non supervisé est une règle d'apprentissage généralement utilisée avec des réseaux autonomes appelés auto-organisateurs. Les algorithmes à base de cette règle d'apprentissage sont alimentés, au cours de la phase d'apprentissage par une base d'apprentissage constituée d'un ensemble de stimuli avec une certaine redondance. Contrairement à la règle d'apprentissage supervisé, il n'existe pas de catégories préalables selon lesquelles les entrées doivent être groupées. L'affectation d'un stimulus sera automatiquement gérée par le réseau afin de générer à la fin de la phase d'apprentissage une sorte de cartographie d'affectation permettant de regrouper les entrées aléatoires de l'environnement externe. Pour cette raison, cette catégorie d'algorithmes d'apprentissage est souvent appelée algorithme de « *clustering* » (regroupement et séparation).

A chaque itération, les neurones dégagent les propriétés statistiques du stimulus pour qu'il soit associé à un circuit neuronal. Le rôle du réseau est d'analyser les relations entre les stimuli et extraire les associations. Au cours de la phase d'apprentissage, les paramètres libres du réseau de neurones seront modifiés afin de fournir une régularité de classement. L'objectif du réseau est donc d'identifier la similarité des entrées présentes sur la base d'apprentissage. Ainsi, la finalité de l'apprentissage est de minimiser la similarité interclasses et maximiser la similarité intra-classes.

Une fois la phase d'apprentissage achevée, le réseau aura la capacité de regrouper les entrées aléatoires selon leurs similarités pendant la phase d'utilisation du système. Bien que tout le traitement semble autonome pour les systèmes à base de règle d'apprentissage non supervisé, pour certaines applications, une opération d'étiquetage parait obligatoire au cours de la transition entre la phase d'apprentissage et celle de décision ou de rappel. Cette dernière commencera par la présentation d'un ensemble de stimuli appelé base d'essai. Cette base est composée des couples (stimulus, étiquette). A la fin de l'opération d'étiquetage, chaque neurone sur la couche de sortie sera étiqueté une classe. C'est une sorte de supervision de la règle dite non supervisée. Toutefois elle n'est nécessaire que pour certaines applications.

Les algorithmes à base de règle d'apprentissage non supervisé sont souvent utilisés pour des fonctions de clustering (règles d'association, reconnaissance de formes, segmentation d'images...). D'autre part, ils ont montré aussi leurs performances pour d'autres domaines d'application, parmi lesquels : la réduction de dimension de données, le changement d'échelle, la correction de données corrompues, la résistance au bruit et aux anomalies, etc... Parmi les algorithmes d'apprentissage non supervisé les plus connus, on trouve la règle de *Hebb* [25], la règle *K-moyenne* [46] et la règle des *mémoires associatives* proposées par *Kohonen en 1977* [47] sur laquelle il a basé

son célèbre réseau de neurones «*la carte auto-organisatrice*» en *1982* [48].

Voici ci-dessous quelques arguments souvent énumérés en faveur de l'approche d'apprentissage non supervisé :

— Garantir l'autonomie des systèmes.
— Faible coût de construction de la base d'apprentissage.
— Ressources matérielles moins couteuses.
— Capacité à l'auto-détection de la nature et la structure des données de l'environnement extérieur.
— Utile pour l'étude des caractéristiques cachées des données.
— Utile pour le pré-traitement des données (élimination du bruit, réduction de la taille des données, classification, etc...).

1.7 Modèles des réseaux de neurones artificiels

Depuis l'apparition des réseaux de neurones artificiels, plusieurs modèles neuronaux ont été proposés. Chaque modèle est caractérisé par sa règle d'apprentissage, les fonctions intra-neuronales (fonction d'activation) et l'architecture des connexions inter-neuronales. Un modèle neuronal à base d'un seul neurone est rarement utilisé. En fait, dans son état isolé, un neurone ne peut offrir de fonction intéressante. Autrement dit, un modèle neuronal est qualifié par les interconnexions entres ces unités élémentaires. Cette section présente quelques modèles de réseaux de neurones artificiels ainsi que les principaux critères de choix des modèles adoptés.

1.7.1 Perceptron

Le *Perceptron Multi Couches* (*PMC*), en anglais *Multi-Layers Perceptron* (*MLP*), est le réseau de neurones le plus utilisé pour des fonctions d'approximation, de diagnostic et de prédiction [49]. Son architecture est conçue de manière que le flux de données se propage dans un seul sens (des entrées vers la ou les sorties) en passant par les couches intermédiaires du réseau. Ce modèle utilise souvent une règle d'apprentissage supervisé. L'algorithme d'apprentissage le plus utilisé ou employé par *MLP* est la retro-propagation du gradient où, la propagation des informations s'effectue dans le sens inverse pour l'adaptation des paramètres libres.

i) Perceptron simple

Un cas particulier des *MLP* est le perceptron simple. Tel qu'il est présenté sur la figure 4, le perceptron simple est une architecture composée d'une seule couche de neurones dont la fonction d'activation est de type « *pas unitaire* » [50]. Tous les neurones du perceptron simple sont alimentés par *D éléments* de la couche d'entrée, où *D* est également la dimension du vecteur de poids $\overrightarrow{W}$ associé au neurone. Les entrées seront traitées par les neurones et présentées à la fonction d'activation de chaque neurone. Cette fonction déterminera l'état d'activation du neurone et

présentera les résultats sur la sortie *S*.

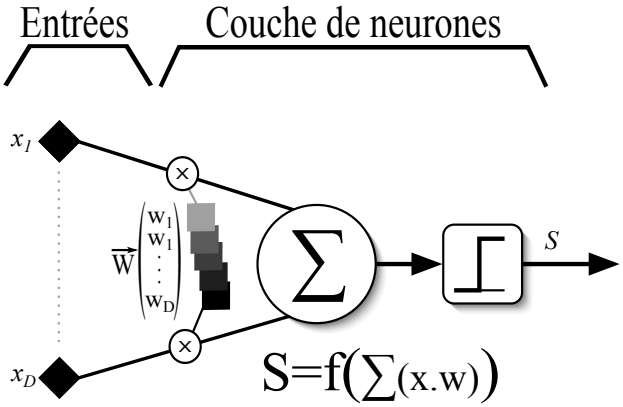


FIGURE *4 – Schéma d'un Perceptron Simple*

Malgré la simplicité de cette architecture et de l'algorithme utilisé, le perceptron simple s'avère très utile pour des fonctions linéaires de classement. Mais les difficultés apparaissent si deux classes sont linéairement non séparables. Dans ce cas, il faut ajouter d'autres couches de neurones. Donc, le perceptron multicouche s'impose comme solution.

ii) Perceptron multicouches

Un *Perceptron Multi Couches* (*PMC*) est un assemblage de plusieurs perceptrons simples placés sous forme de N couches concaténées les unes aux autres (voir figure 5) [51]. Suivant cette mise en œuvre, le flux de données se propage dans un seul sens (entrée vers sortie). Les neurones d'une couche sont alimentés par les sorties de la couche adjacente. Selon leur disposition, on trouve deux types de couche :

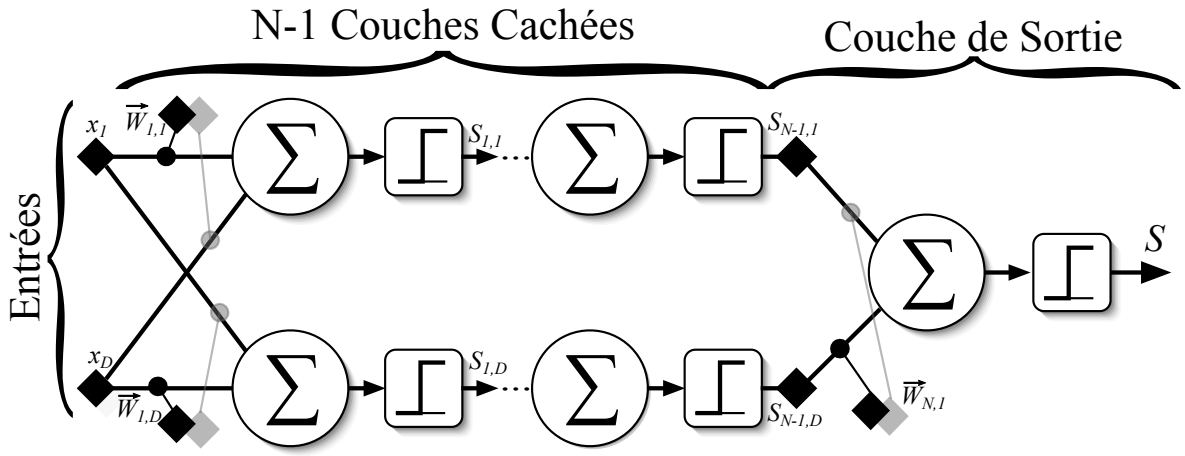


FIGURE *5 – Schéma du Perceptron Multicouche*

— La couche de sortie appelée souvent couche de décision. C'est la couche à l'extrémité droite. Sur cette couche, les résultats finaux de traitement seront récupérés. Le nombre de neurones sur la couche de sortie dépend donc des résultats désirés. Cette couche peut être

présentée par un perceptron simple. Entre autre, elle peut être présentée par un autre modèle neuronal appelé *ADALINE* [41]. Ce modèle à la même architecture que le perceptron simple sauf que les neurones utilisés par ce modèle, ont une fonction d'activation linéaire.

— Les couches cachées ou couches de prétraitement. Ces couches sont placées entre les entrées et la couche de sortie. Leur rôle est de préparer les données en utilisant principalement des fonctions d'activation non-linéaires dans leurs neurones pour les présenter à la couche de sortie. Pour cela, ces couches doivent être de type perceptron simple et non pas de type *ADALINE*. Car, en combinant plusieurs fonctions linéaires on obtient une fonction linéaire. Le nombre de neurones sur chaque couche cachée ainsi que le nombre de couches cachées dépendent du traitement à effectuer ainsi que du comportement du réseau souhaité.

1.7.2 Réseaux de neurones récurrents

Contrairement au *MLP*, avec modèle neuronal récurrent, le sens de propagation du flux de données est aléatoire. Les sorties d'une couche peuvent être utilisées comme entrées de n'importe quelle autre couche et pas uniquement de la couche suivante. De même, la rétro-propagation des données est également possible [52]. Ce modèle neuronal utilise souvent une règle d'apprentissage non supervisé. La figure 6 présente l'exemple du modèle *Fuzzy Adaptive Resonance Theory* (*Fuzzy ART*) [53]. Avec, $A(t)$ résultats du pré-traitement des données à l'entrée, $\overrightarrow{W}$ les poids des neurones des couches cachées et ρ paramètres de précision.

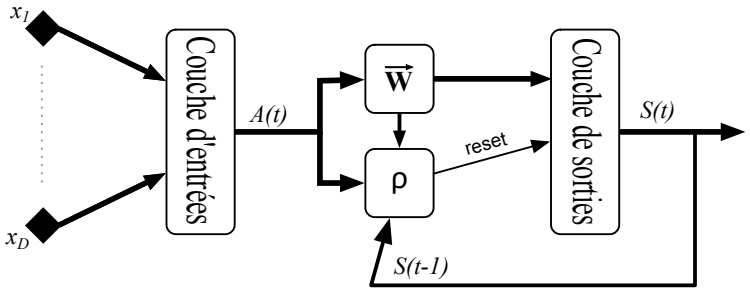


FIGURE *6 – Schéma du modèle Fuzzy ART*

Les réseaux de neurones récurrents sont très répandus pour la réalisation de fonctions de reconnaissance de formes ou de changement d'échelle. Bien qu'ils soient assez complexes, les réseaux récurrents proposent de bonnes performances avec un cycle d'apprentissage court par rapport à leurs homologues *MLP*.

1.7.3 Réseaux de neurones compétitifs

Les réseaux de neurones artificiels compétitifs possèdent généralement une architecture composée d'une couche de neurones massivement inter-connectés. La figure 7 présente l'architecture générale d'un réseau de neurones artificiels compétitif. Le traitement au niveau du réseau passe

par deux étapes [54] : l'étape d'extraction de la relation entre le stimulus et chaque neurone du réseau, ensuite l'étape de compétition inter-neuronale.

Chaque neurone est caractérisé par son vecteur de poids de dimension D (la même dimension pour tous les neurones). Au niveau de la couche d'entrée, un vecteur d'entrée, de la même dimension que celle du vecteur de poids, sera distribué à tous les neurones du réseau. Pour chaque vecteur d'entrée, les neurones entrent en compétition pour sélectionner le neurone gagnant ou les neurones gagnants pour le stimulus injecté. Un neurone gagnant est identifié selon la compétition effectuée. En général, ce neurone est identifié en fonction de sa ressemblance au vecteur d'entrée du stimulus.

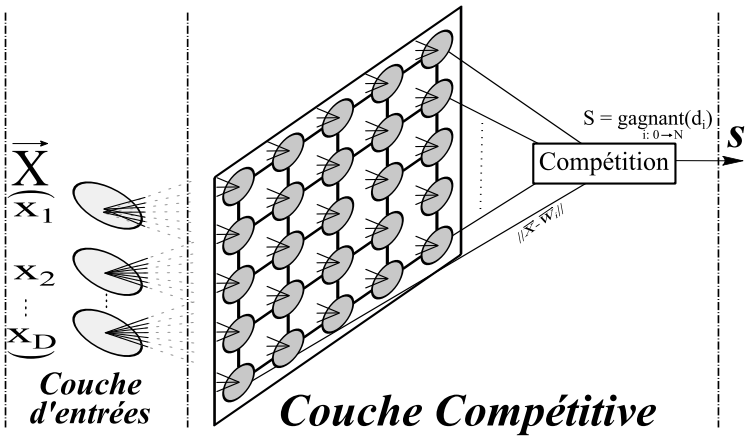


FIGURE 7 – *Schéma du Réseau de neurones Compétitif*

La règle d'apprentissage la plus adaptée pour ces modèles neuronaux est la règle non supervisée [55–57]. Exceptionnellement, des algorithmes d'apprentissage supervisé peuvent être utilisés [58, 59]. Au cours de la phase d'apprentissage, le réseau adapte les éléments de ses vecteurs de poids par rapport au stimulus injecté. L'adaptation des vecteurs de poids peut suivre deux règles :

Règle d'adaptation locale : Dans ce type d'adaptation seul le neurone gagnant voit son vecteur de poids modifié. Cette adaptation permet au neurone sélectionné d'accroitre ses chances d'activation pour des stimuli similaires [60]. Pour les modèles avec une règle d'apprentissage supervisé, l'adaptation sera effectuée afin de minimiser la reproduction d'erreur. En cas de non correspondance entre la classe du neurone et l'étiquette du stimulus, une modification des poids des neurones gagnants éloignera ces derniers des stimuli d'entrée. Ou à l'inverse, l'adaptation rapprochera les poids afin d'augmenter la chance de sélection du neurone pour des stimuli similaires.

Règle d'adaptation globale : Contrairement à la règle d'adaptation locale, l'approche globale permet la modification des éléments des vecteurs de poids au niveau de tous les neurones du réseau. En suivant une règle de voisinage, les vecteurs de poids seront adaptés au vecteur de stimulus injecté. En général, l'apprentissage génère une cartographie regroupant plusieurs

neurones pour une même classe de stimulus d'entrée *C*, ceci améliorant le « *Clustering* » du réseau [61].

1.7.4 Réseaux de neurones évolutifs

Les réseaux de neurones évolutifs, souvent appelés réseaux de neurones constructifs, sont des réseaux de type compétitifs [62]. Initialement, la couche destinée au traitement est composée d'un nombre minimal de neurones, comme présenté sur la figure 8(a). Au cours de la phase d'apprentissage (voir figure 8(b)), les paramètres libres du réseau ainsi que les connexions inter-neuronaux seront adaptés à la base d'apprentissage composée des stimuli représentant la majorité des exemples de l'espace d'entrée. A un certain niveau d'apprentissage, tel qu'illustré sur la figure 8(c), de nouveaux neurones sont ajoutés selon les besoins de l'application [63].

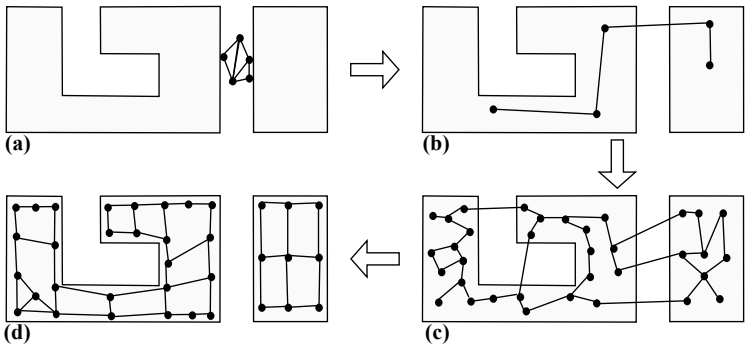


FIGURE *8 – Principe des réseaux de neurones évolutifs. (a) : Etat initial, (b) : Adaptation, (c) : Évolution du réseau, (d) : Adaptation finale.*

A la fin de la phase d'apprentissage, le réseau est composé d'un nombre optimal de neurones (figure 8(d)), étant la fonction des paramètres applicatifs et critères d'arrêt déterminés par l'utilisateur avant la phase d'apprentissage. Les paramètres libres ainsi que les interconnexions seront adaptés au comportement idéal du réseau de neurones pour satisfaire les besoins de l'application.

Ce modèle neuronal adopte une règle d'apprentissage non supervisé. Parmi les réseaux de neurones les plus répandus avec ce modèle, on cite le réseau *Growing Neural Gas* (*GNG*) [64]. Ce réseau de neurones est caractérisé par sa précision ainsi que son temps d'apprentissage réduit. Ils sont généralement utilisés pour des fonctions de reconnaissance de formes, de classification, etc… ou pour aider les systèmes à la résistance aux bruits et aux anomalies.

1.7.5 Les réseaux de Kohonen

Les réseaux de Kohonen sont classés dans la catégorie des réseaux de neurones compétitifs. A partir de son approche de mémoires associatives, *Teuvo Kohonen* a proposé la *carte auto-organisatrice SOM en 1982* [55]. Après le succès de sa carte auto-organisatrice, *Teuvo Kohonen* a proposé un modèle adoptant un algorithme d'apprentissage supervisé. Ce nouveau modèle est appelé *Learning Vector Quantization* (*LVQ*) [60]. Dans cette version, l'algorithme d'apprentissage

suit une règle d'adaptation locale. L'objectif de cet algorithme consiste à rapprocher le neurone gagnant de la position cartographique du stimulus en cas de succès. Dans le cas contraire, en cas d'erreur, l'objectif est d'écarter le neurone en question de la zone des stimuli similaires de l'espace d'entrée.

Les réseaux de Kohonen sont généralement utilisés pour des applications de classification. Dans ce cadre, le modèle *LVQ* a été présenté afin de faciliter l'opération d'étiquetage de la carte auto-organisatrice. D'autre part, les réseaux de Kohonen ont montré leurs performances pour des applications de reconnaissance de forme, de réduction de taille des données ainsi ils sont caractèrisés par une certaine robustesse aux données bruitées et/ou corrompues.

1.7.6 Choix de réseau de neurones artificiels

Il existe plusieurs modèles de réseau de neurones artificiels. Chaque modèle est caractérisé par son architecture, son traitement et sa règle d'apprentissage. Pour choisir le modèle le plus adapté à une application définie, il faut prendre en compte différents paramètres, parmi lesquels nous citons :

— La fonction désirée (classification, prédiction, diagnostic ou reconnaissance).
— La nature des données à traiter. Ces données peuvent être de nature dynamique, statique ou aléatoire et peuvent avoir différentes formes.
— Ressources matérielles et/ou logicielles disponibles pour l'implémentation du réseau.
— Contraintes temporelles généralement liées à des applications temps réel.
— Les efforts de préparation de la base d'apprentissage ainsi que de la base de tests et validation en cas de besoin.
— Délais d'apprentissage correspondant au temps nécessaire avant de considérer le réseau comme expert et commencer la décision.

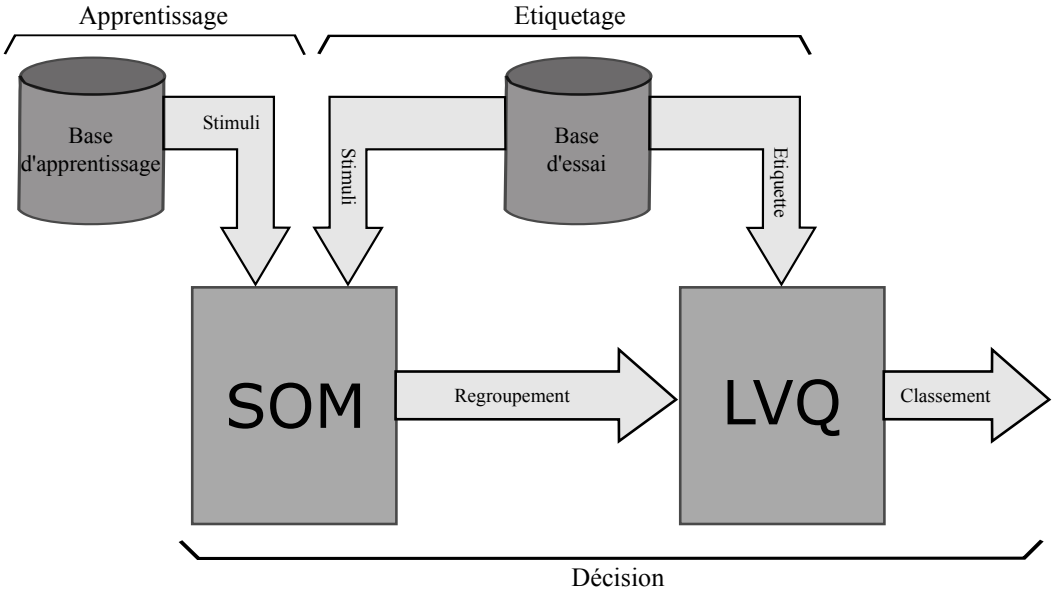


FIGURE 9 – *Couplage de deux modèles neuronaux (SOM + LVQ).*

Certaines applications sont conçues à base de plusieurs réseaux de neurones. Dans ce cas, il faut prendre en compte l'adaptabilité entre les modèles couplés. L'exemple de la figure 9 propose un couplage de deux réseaux de Kohonen (*SOM* et *LVQ*) dans ses deux modèles : supervisé et non supervisé. Ce couplage est généralement utilisé pour des applications de classification ce qui permet l'étiquetage des résultats de la carte auto-organisatrice à base d'une carte *LVQ*.

1.8 La carte auto-organisatrice de Kohonen

Les travaux de recherche effectués au sein de notre laboratoire reposent sur des fonctions de classification et de séparation des données expansives en termes de taille et de redondance. Dans ce cadre, nous avons choisi d'utiliser et de nous focaliser sur des réseaux de neurones compétitifs. Plusieurs modèles neuronaux dans cette catégorie ont été proposés depuis l'introduction des principes d'adaptation neuronale par *Von der Malsburg* [65–68] :

— Tous les neurones situés sur couche compétitive sont en relation directe avec les unités sensorielles représentant la couche d'entrée. Le comportement global du réseau de neurones consiste en une élection d'un neurone gagnant en fonction des différentes caractéristiques des informations interceptées par les unités sensorielles.

— L'adaptation des paramètres libres (au cours de l'apprentissage) doit respecter la distribution topologique des neurones en fonction du voisinage autour du neurone gagnant. Par conséquent, la zone d'inhibition sera distribuée en fonction du voisinage de ce neurone pour des stimuli similaires.

L'élection du neurone gagnant sur ces modèles est effectuée via des formules mathématiques basées sur l'échange d'information entre les unités fonctionnelles via des connexions latérales. Ensuite, l'adaptation des poids des neurones selon leur appartenance à la zone d'activation ou à une zone d'inhibition, est assurée à base d'une fonction de voisinage similaire à la fonction d'interaction latérale appelée « *Chapeau mexicain* ». La figure 10 présente la courbe de cette fonction.

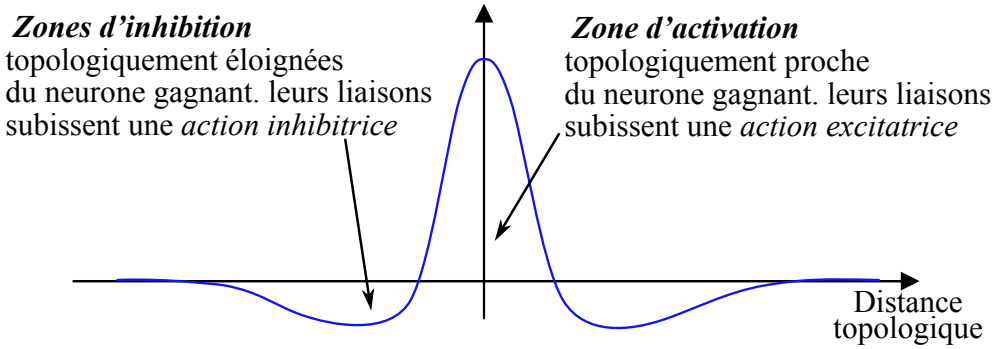


FIGURE *10 – Chapeau Mexicain «* Mexican hat *» : Fonction d'interaction latérale*

De sa part, *Teuvo Kohonen* a proposé en *1982* un algorithme innovant est relativement simple. L'objectif principal de cet algorithme est de faire correspondre les éléments de l'espace d'en-

trée avec les neurones représentant les unités élémentaires de son réseau de neurones artificiels nommé *la carte auto-organisatrice* ou «*Self-Organising Map* (*SOM*)». Pour garantir ce comportement, *Kohonen* propose une simplification algorithmique du modèle neuronal proposé par *Von der Malsburg* [65]. Le principe d'adaptation a été remplacé par un simple procédé non linéaire reposant sur les deux principes précédemment cités.

Avec l'algorithme *SOM*, l'élection du neurone gagnant, via des connexions latérales entre les neurones, a été remplacée par une fonction de proximité entre le stimulus et le neurones. Cette proximité est calculée en fonction des distances Euclidiennes entre le vecteur d'entrée $\overrightarrow{X}$ du stimulus et les vecteurs de poids $\overrightarrow{W}$ des neurones. Le neurone gagnant est celui possédant la distance minimale. D'autre part, la formation de la *zone d'activation* et des *zones d'inhibition*, basée sur la fonction de « *chapeau mexicain* », a été simplifiée dans une première version [48,60] par une fonction de voisinage «*tout-ou-rien*» à base d'un *seuil de proximité* topologique. Ensuite dans la dernière version [55], la fonction de voisinage suit une *dégradation gaussienne* en fonction du facteur de voisinage par rapport au neurone gagnant (voir figure 11).

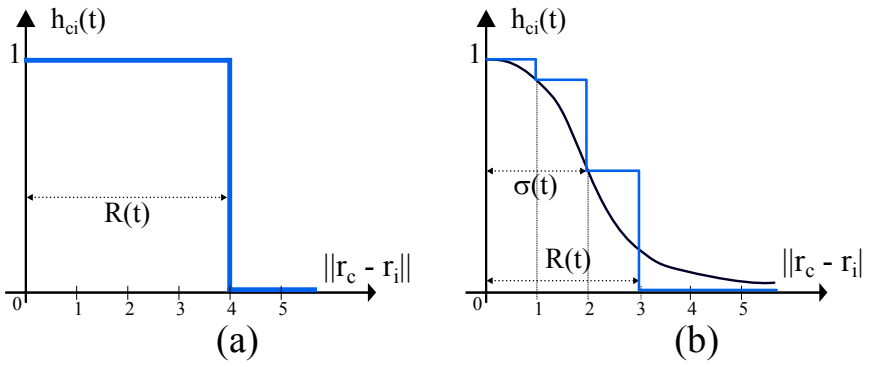


FIGURE *11 – Fonction de voisinage de la carte auto-organisatrice : (a) fonction* tout-ou-rien *; (b) fonction* Gaussienne

Par conséquent, l'optimisation de l'objectif principal qui consiste à réaliser une projection de l'environnement externe sur la couche compétitive du *SOM* est garantie par deux propriétés de l'algorithme :

— Les données en entrée ayant un certain niveau de similarité seront assignées à des neurones proches en termes de voisinage topologique.

— Les données des régions très denses dans l'espace d'entrée (les régions où la fréquence d'injection de leurs données dans la carte est plus élevée) doivent être assignées à un ensemble composé d'un nombre de neurones plus important que les ensembles assignés aux données des régions peu denses.

Une caractéristique générale peut englober les principes et les propriétés sur lesquels *Kohonen* a fondé son algorithme *SOM* de la *carte auto-organisatrice*. En effet, la *carte auto-organisatrice* est un modèle neuronal permettant de créer des relations topologiques des données de l'environnement externe, en se basant uniquement sur les vecteurs d'entrée présentés à la carte, étant les représentants principaux des caractéristiques de l'environnement externe dans lequel le système

évolue.

1.8.1 Architecture de la carte auto-organisatrice

L'architecture d'une carte auto-organisatrice de Kohonen, tel qu'elle est présentée sur la figure 12, est composée de trois couches : une *couche d'entrée*, une *couche de sortie* et la *couche compétitive* souvent appelée *couche de Kohonen*. Sur cette dernière, on trouve un ensemble de neurones inter-connectés via des liaisons de voisinage. Chaque neurone sur la carte est caractérisé par son *vecteur de poids* ou *vecteur de pondération*. En termes de la projection de l'environnement externe sur la carte, le vecteur de poids représente la position du neurone sur l'espace d'entrée. Par conséquent, la dimension du vecteur de poids est égale à la dimension de l'espace d'entrée. Les échantillons prélevés de cet espace de données seront présentés sur la couche d'entrée élément par élément. Ensuite, ils seront envoyés vers tous les neurones de la couche compétitive. Les distances calculées par ces derniers seront envoyées à leur tour vers le comparateur de la couche de sortie pour déterminer la classe associée à l'échantillon sur cette couche de sortie (voir la figure 12).

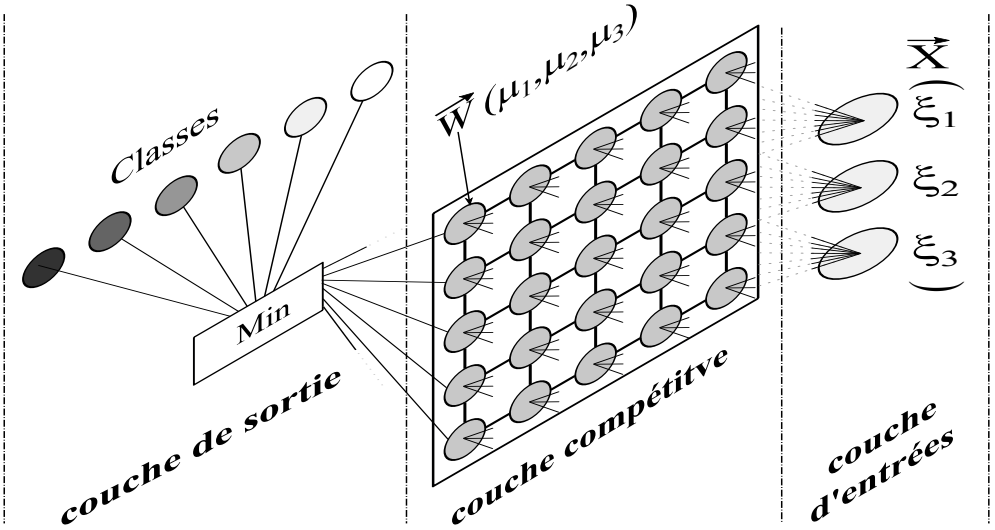


FIGURE *12 – Architecture de la carte auto-organisatrice de Kohonen*

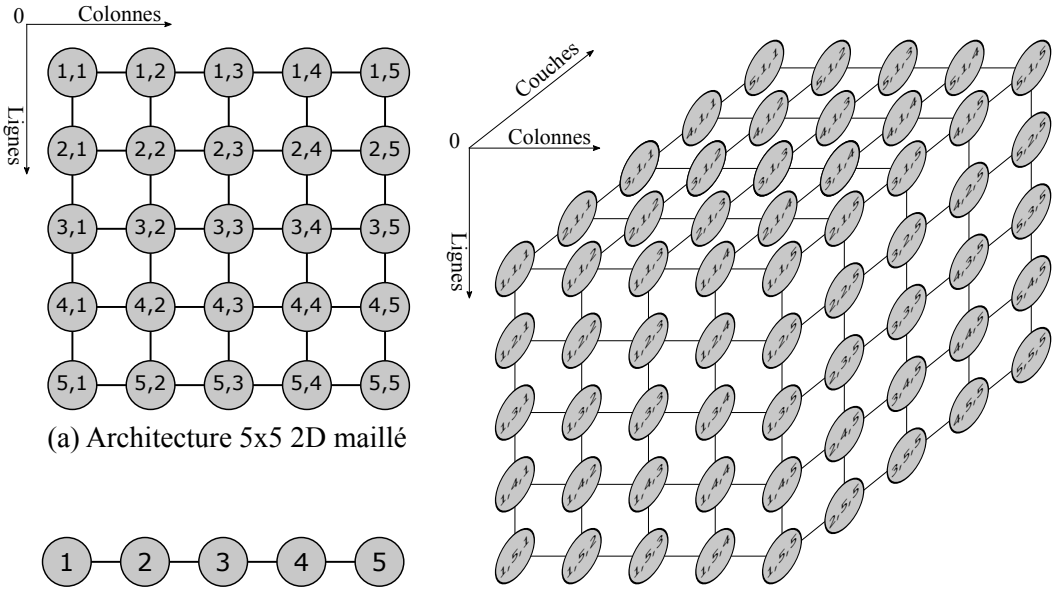
Au cours de la phase d'apprentissage de la carte auto-organisatrice, les échantillons en entrée sont injectés d'une façon itérative. Chaque échantillon est traité à part par le réseau sans attendre un retour de l'extérieur. On parle donc du principe de l'apprentissage non-supervisé. Les échantillons de l'espace d'entrée sont sélectionnés d'une manière aléatoire au cours de cette phase et permettent de représenter statistiquement l'espace d'entrée. L'objectif de la carte auto-organisatrice est d'isoler et retrouver les relations topologiques des vecteurs d'entrée en les projetant sur la couche compétitive de la carte auto-organisatrice [69]. La création de cette projection s'effectue par l'adaptation des vecteurs de poids du neurone élu vainqueur ainsi que ses voisins. Cela permet de créer des *clusters* représentant la majorité des classes existantes avec une densité

neuronale relativement similaire à celle de l'espace d'entrée, tout en gardant la même topologie de distribution de ces neurones sur la carte. Les neurones appartenant au même cluster ont les poids très proches même si parfois physiquement peuvent être éloignés dans la carte des neurones.

Chaque neurone est identifié par sa position topologique sur la carte. A base de cette identité, on peut calculer le taux de voisinage entre les neurones selon leur topologie de voisinage (connexions inter-neuronales) ainsi que la topologie de distribution des neurones sur la couche compétitive.

i) Topologie de distribution de la carte de Kohonen

La *topologie de distribution* des neurones sur la carte auto-organisatrice désigne la façon à base de laquelle les neurones sont distribués sur la couche compétitive. Dans la littérature, on distingue trois topologies principales présentées sur la figure 13 : *distribution linéaire*, *distribution en 2D* et *distribution en 3D*.



(a) Architecture 5x5 2D maillé

(b) Architecture Linéaire

(c) Architecture 5x5x5 3D

FIGURE *13 – Différentes topologies de la carte auto-organisatrice*

Les *distributions linéaires* de la carte *SOM* [1, 70] sont généralement utilisées pour des fonctions de classification simples. En général, le nombre de classes existantes dans l'espace d'entrée est relativement réduit avec une densité équilibrée de données. Dans cette architecture, la couche de Kohonen (couche compétitive), telle qu'elle est présentée sur la figure 13(b), est composée d'une ligne de neurones avec une liaison de voisinage directe entre les neurones adjacents. Chaque neurone est identifié par sa position sur la ligne à partir d'une position de référence. En général, le neurone situé à l'extrémité gauche de la ligne a l'indice le plus faible. Cet indice s'incrémente en se déplaçant d'un neurone à un autre vers la droite.

Cette architecture est relativement simple à implémenter, mais, limite le parallélisme des communications du réseau de neurones. Avec des implémentations logicielles, où le parallélisme de

l'algorithme n'est pas un atout, cette distribution peut être une solution pour des fonctions de classification sur des espaces linéaires [70] ou des applications de détection de contour [71]. D'un autre coté, pour des implémentations matérielles cette distribution peut limiter les performances des architectures classiques de la carte *SOM* [72]. En contrepartie, cette topologie peut être exploitée pour des améliorations des architectures matérielles, en changeant la démarche de traitement des données en entrée. Par exemple, *Manolakos* et *al.* dans [1], ont proposé une architecture matérielle linéaire sur laquelle les trois couches de l'architecture de la carte auto-organisatrice présentées auparavant sont fusionnées. Dans cette architecture présentée sur la figure 14, le traitement global a été synchronisé avec une *propagation systolique* des éléments du vecteur d'entrée sur les unités de traitement des éléments du vecteur de poids qui ont été distribués sur une grille en 2D. D'où, les distances sont calculées et comparées avec une propagation systolique sur une architecture totalement distribuée.

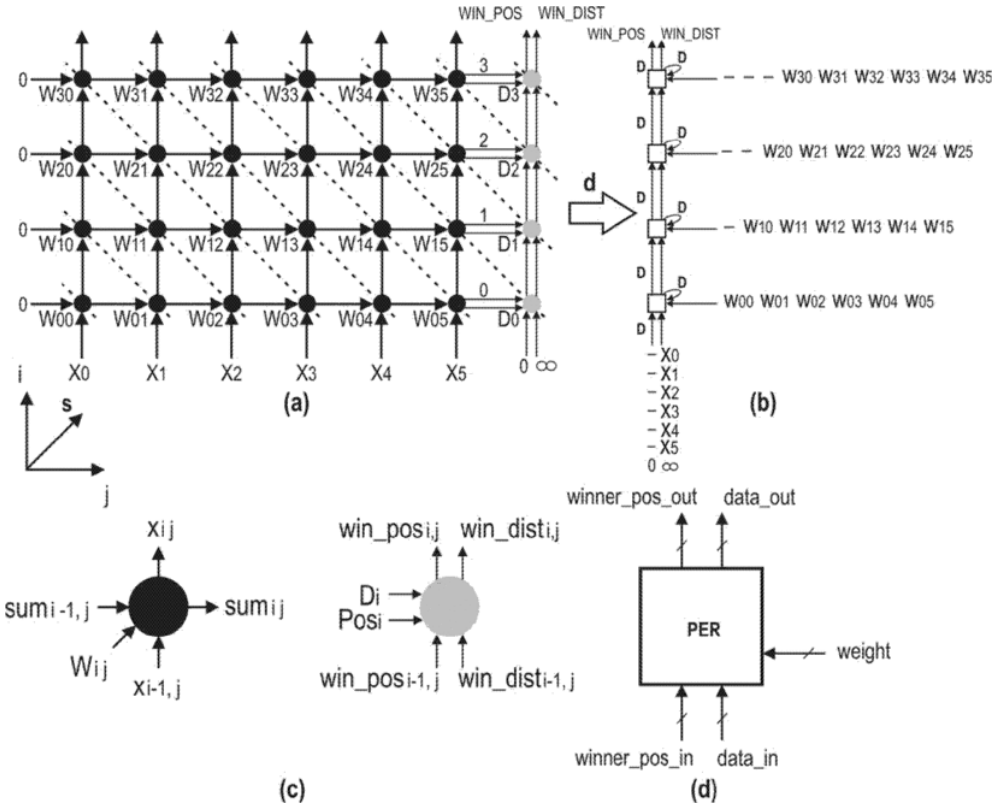


FIGURE *14 – Architecture linéaire systolique distribuée : (a) Graphe de dépendance de données en entrée ; (b) graphe de flux de données (c) Les modules de calcul de la distance (d) Comparateur linéairement distribué [1]*

D'autre part, une architecture à *distribution 3D* est présentée sur la figure 13(c). Dans cette architecture, les neurones de la couche compétitive sont distribués sur un plan à trois dimensions : Couche, Ligne et Colonne. Chaque couche est composée d'une grille de neurones. Sur cette grille, les neurones sont distribués selon une topologie 2D maillée. Chaque neurone de la carte est identifié par trois indices (*c, l* et *k*) : l'indice *c* de la couche à laquelle il appartient, l'indice *l* de la ligne et l'indice *k* de la colonne. Par conséquent, l'identité du neurone est souvent présentée

par ces trois valeurs. Chaque neurone est connecté directement à ses voisins adjacents en termes de ligne et colonne d'une part, et aux neurones qui sont situés sur des couches adjacentes à la sienne et qui ont les mêmes indices *l* et *k*.

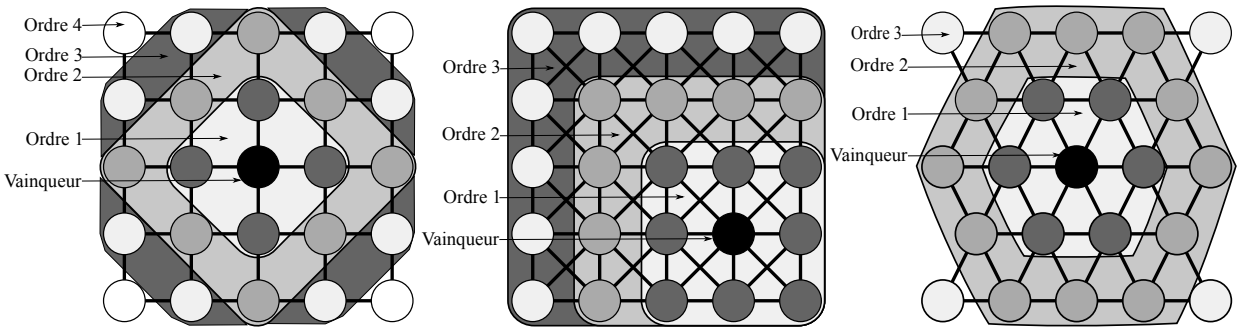
Dans la littérature, il existe plusieurs travaux d'implémentations des réseaux de Kohonen en 3D. Les approches proposées sont généralement utilisées pour améliorer le traitement des données présentées sur des espaces d'entrée à *3-dimensions*, dans des domaines de la réalité augmentée [73] ou de reconstructions d'image en 3D [74]. Cette distribution permet de réaliser une meilleure projection de la topologie de l'environnement réel. Les implémentations matérielles adoptant cette distribution, cherchent à améliorer les performances temporelles ainsi que les performances de précision. Dans [75], *N. Sudha* a proposé une architecture 3D parallèle de type *Single Instruction Multiple Data* (*SIMD*) de la carte auto-organisatrice. Cette architecture a été utilisée pour une application de quantification d'image couleur. L'idée est de décomposer les couleurs RVB (rouge, vert et bleu) pour ajouter une dimension à la topologie de projection.

Les architectures à *distribution 2D* est l'architecture la plus utilisée pour les implémentations de la carte auto-organisatrice. Cette architecture telle qu'elle est présentée sur la figure 13(a), est composée d'un ensemble de neurones distribués sur un plan à deux dimensions. Chaque neurone est identifié par sa position sur le plan. Les identités des neurones peuvent être présentées sous deux formes : indice simple ou indice double. Les identifiants simples sont présentés par un chiffre [72, 76]. Le neurone placé en haut et à gauche de la grille possède l'identité *1*. Il est la référence du compteur des indices. L'indice s'incrémente en se déplaçant d'une colonne à une autre, ligne par ligne. Le calcul du taux voisinage avec cette approche s'effectue à base d'une formule de division et modulo. Les identifiants à double indices sont présentés par un couple de d'indice *(l, k)* [77, 78]. Ces indices désignent la position du neurone en termes de ligne (indice *l*) et en termes de colonne (indice *k*). Cette présentation permet de simplifier le calcul du taux de voisinage. En effet, le taux de voisinage est calculé ainsi en fonction des liaisons entre les neurones. Sur les architectures à *distribution 2D*, on peut trouver différents types de liaisons inter-neuronales. A base de ces liaisons on peut identifier la topologie de voisinage utilisée.

D'autre part, il existe, dans la littérature d'autres types de topologies de distribution rarement utilisées. Ces approches sont utilisées pour des applications dédiées ou pour des contraintes matérielles fortes. Parmi ces topologies, on cite : les architectures à *topologie mixte* qui sont composées d'une combinaison de *topologie linéaire* et *2D* [79]. Les architectures à base de *topologie hiérarchique* sont composées des neurones reliés par des liens hiérarchiques sur différents niveaux [80]. Sur ce type de topologie, un neurone appartenant à un niveau est lié uniquement à un des neurones du niveau supérieur et à un ensemble de neurones sur le niveau inférieur. Par conséquent, la topologie de distribution de ce type d'architecture prend la forme d'un arbre hiérarchique. Enfin, on trouve également dans la littérature des *topologies irrégulières*, où les neurones sont distribués d'une façon irrégulière sur la couche compétitive.

ii) Topologie de voisinage de la carte de Kohonen

La *topologie de voisinage* sur une carte auto-organisatrice désigne la manière d'interconnexion des neurones sur la couche compétitive. A base de cette topologie, on peut identifier les liens de voisinage entre les neurones pour calculer le taux de voisinage ainsi que l'ordre de voisinage limité par le rayon de voisinage au cours de la phase d'apprentissage de la carte [81]. Dans cette section, nous nous intéressons à des topologies de voisinage utilisées dans les architectures 2D. La figure 15 présente trois types de topologies de voisinage : voisinage *en losange*, voisinage *rectangulaire* et voisinage *hexagonal*.



(a) Voisinage en losange (a) Voisinage rectangulaire (a) Voisinage hexagonal

FIGURE *15 – Topologies de voisinage de la carte auto-organisatrice*

La topologie de *Voisinage en losange*, souvent appelée topologie de *voisinage adjacent* [82], consiste à interconnecter chaque neurone à ses voisins adjacents sur la ligne et la colonne auxquelles il appartient. Avec cette topologie, l'ordre de voisinage est distribué sous forme des losanges croissants. Le nombre maximal de neurones de premier ordre de voisinage est de quatre voisins. Ce nombre se multiplie par deux en passant à l'ordre de voisinage suivant.

Dans une topologie de *voisinage rectangulaire* [83], chaque neurone est connecté non seulement à ses voisins adjacents, mais aussi à ces voisins les plus proches en diagonal. D'où l'ordre de voisinage est présenté sous forme de rectangle. Sur le premier ordre, on trouve un ensemble de huit neurones qui se multiplie en passant à un ordre de voisinage plus éloigné. Par conséquent, le nombre de neurones concernés par l'adaptation de leurs vecteurs de poids avec cette topologie est multiplié fois deux par rapport à celui de la topologie de voisinage adjacent.

Une troisième topologie de voisinage est proposée par *Kolsa* dans [84]. Cette topologie dite *hexagonale* ou *voisinage triangulaire*, permet d'obtenir un nombre de neurones intermédiaire entre les deux topologies de voisinage précédemment citées. Cette topologie est réalisée à base des liaisons triangulaires entre un neurone et ses voisins sur une ligne adjacente. D'autre part, un neurone est connecté directement à ses voisins adjacents en termes de colonne.

1.8.2 Algorithme Self-Organizing Map (SOM) de Kohonen

L'algorithme *SOM* se base sur une approche d'apprentissage compétitif non-supervisé [85]. L'objectif est d'adapter les paramètres libres de la carte auto-organisatrice, représentés par les

vecteurs de poids de ses neurones, à la topologie de l'espace d'entrée, tout en gardant la disposition de voisinage des neurones définie au préalable. Cela s'effectue à base d'un algorithme itératif non supervisé. Les étapes de cet algorithme sont les suivantes :

1. Initialiser les paramètres de la carte auto-organisatrice : les vecteurs de poids $W_{l,k}$, le rayon de voisinage $R(t)$ et le taux d'apprentissage $\boldsymbol{\alpha}(t)$.
2. Envoyer un vecteur d'entrée, associé à un échantillon, sur la couche d'entrée.
3. Calculer la distance entre le vecteur d'entrée et les vecteurs de poids de chaque neurone sur la couche compétitive.
4. Élire le neurone gagnant correspondant au neurone ayant la distance la plus petite.
5. Mettre à jour les vecteurs de poids du neurone gagnant ainsi que ceux de ses voisins d'ordre inférieur au rayon de voisinage.
6. Procéder à la réduction du rayon de voisinage $R(t)$ et du taux d'apprentissage $\boldsymbol{\alpha}(t)$. Ensuite, reprendre l'algorithme à partir de l'étape (2).

Après plusieurs itérations des étapes de *2* à *6*, la carte s'auto-adapte pour converger à une présentation réduite de l'espace d'entrée. Cette présentation tend à conserver la densité de l'espace d'entrée, avec une marge d'erreur tolérée. Cela s'explique par rapport à la taille de la couche compétitive qui est réduite par rapport à la taille de données de l'espace réel des entrée. En effet, c'est une des propriétés de la carte auto-organisatrice qui consiste à la réduction de la taille des données de l'espace d'entrée.

On remarque que le déroulement de *l'algorithme de Kohonen*, tel qu'il est présenté sur l'algorithme 1, passe par trois phases : une phase d'*initialisation*, une phase d'*apprentissage* et enfin une phase de *rappel*.

i) Phase d'initialisation

La *phase d'initialisation* est exécutée au départ. Cette phase consiste à initialiser les valeurs des éléments de vecteur de poids de chaque neurone sur le réseau. Cette initialisation peut influencer le temps de convergence de la couche compétitive vers la topologie de l'espace d'entrée. dans la littérature, on trouve principalement trois types d'initialisation [79, 86, 87] : initialisation par principaux composants de la base d'apprentissage "*Principal Components Initialization* (*PCI*) ", initialisation aléatoire "*Random Initialization* (*RI*) "et initialisation linéaire "*Linear Initialization* (*LI*) ". L'initialisation *PCI* consiste à initialiser chaque vecteur de poids par un vecteur choisi aléatoirement de la base d'apprentissage. L'initialisation *RI* consiste à initialiser les vecteurs des poids par des valeurs calculées à base d'une fonction linéaire d'une manière aléatoire. L'initialisation *LI* consiste à initialiser le vecteur de poids de chaque neurone avec des valeurs calculées d'une façon linéaire en fonction de sa position sur la grille. De cette façon les vecteurs seront distribués d'une façon ordonnée sur la grille et avec une densité équilibrée. Une observation présentée dans [88], propose une méthode d'initialisation qui consiste à recopier les vecteurs de poids d'une carte déjà

Algorithme 1 Algorithme Self-Organizing Map (*SOM*) de Kohonen

Initialisation :

1. Initialiser les vecteurs de poids de chaque neurone.

2. Initialiser des paramètres d'apprentissage.

$$t = 0\,;\, R(0) = major(\frac{(K+L)}{2})\,;\, \boldsymbol{\alpha}(0) \simeq 1 \tag{4}$$

Apprentissage : *Compétition :*

3. Sélectionner aléatoirement un échantillon de la base d'apprentissage.

4. Présenter le vecteur d'entrée $\overrightarrow{\mathrm{X}}$ sur la couche d'entrée.

$$\tilde{X} = (\xi_1, \xi_2, \ldots, \xi_D) \in \mathfrak{Re}^D \tag{5}$$

5. Calculer la distance Euclidienne pour chaque neurone.

$$\left\|\overrightarrow{\mathrm{X}} - \overrightarrow{\mathrm{W_i}}\right\| = \sqrt{(\xi_1 - \mu_{i,1})^2 + (\xi_2 - \mu_{i,2})^2 + \cdots + (\xi_D - \mu_{i,D})^2} \tag{6}$$

6. Chercher l'identité du neurone gagnant.

$$C = \arg\min_i = \left\|\overrightarrow{\mathrm{X}} - \overrightarrow{\mathrm{W_i}}\right\| \tag{7}$$

Adaptation :

7. Calculer le taux de voisinage.

$$h_{C,i}(t) = \exp(-\frac{\left\|\overrightarrow{\mathrm{r_c}} - \overrightarrow{\mathrm{r_i}}\right\|}{2\sigma^2(t)}) \tag{8}$$

8. Mettre à jour le vecteur de poids.

$$\overrightarrow{\mathrm{W_i}}(t+1) = \begin{cases} \overrightarrow{\mathrm{W_i}}(\mathrm{t}) + \boldsymbol{\alpha}(\mathrm{t}) \times \mathrm{h_{C,i}(t)} \times (\overrightarrow{\mathrm{X}} - \overrightarrow{\mathrm{W_i}}(\mathrm{t})) & \mathrm{si} \left\|\overrightarrow{\mathrm{r_c}} - \overrightarrow{\mathrm{r_i}}\right\| \leq \mathrm{R(t)} \\ \overrightarrow{\mathrm{W_i}}(\mathrm{t}) & \mathrm{sinon} \end{cases} \tag{9}$$

9. Mettre à jour les paramètres d'apprentissage.

$$t = t+1\,;\, \boldsymbol{\alpha}(t+1) = \boldsymbol{\alpha}(t) - \tfrac{t}{500 \times L \times K}$$

$$R(t+1) = R(t) - major(\frac{t}{50 \times L \times K}) \tag{10}$$

Rappel : *Compétition :*

10. Présenter un vecteur $\overrightarrow{\mathrm{X}}$ de l'espace réel sur la couche d'entrée.

11. Calculer la distance Euclidienne pour chaque neurone.

12. Chercher l'identité du neurone gagnant.

Décision :

13. Identifier la classe du vecteur sur le cluster de la couche de sortie

entrainée, même si l'ensemble d'entrée est différent. Cette approche permet d'accélérer la phase d'apprentissage.

D'autre part, la phase d'initialisation consiste également à fixer les paramètres globaux d'apprentissage de la carte auto-organisatrice. On trouve généralement deux paramètres principaux : le rayon de voisinage $R(t)$ et le taux d'apprentissage $\boldsymbol{\alpha}(t)$.

Le *rayon de voisinage* $R(t)$ représente *l'ordre de voisinage* du neurone le plus éloigné du neurone gagnant qui sera concerné par l'adaptation. De plus, pendant les premières itérations

d'apprentissage, seulement les neurones appartenant au cercle d'apprentissage de rayon *R* et de centre *C* (où *C* est l'identité du neurone gagnant) seront concernés par la mise à jour de leurs vecteurs de poids. Ce paramètre d'apprentissage est souvent initialisé par une valeur entière supérieure à $\frac{D}{2}$ (où *D* est le diamètre de la grille de neurones $D = \frac{L+K}{2}$ avec *L* nombre de lignes et *K* nombre de colonnes) [55]. Le rayon de voisinage se décrémente au cours de l'apprentissage pour améliorer la reproduction de la topologie de l'espace d'entrée. En général, la valeur de $R(t)$ doit se décrémenter, après un nombre d'itérations égal à *30 à 50 fois* le nombre de neurones sur la carte, jusqu'à ce qu'elle soit réduite à *0*. Il s'agit d'une valeur empirique proposée par Kohonen dans ses premiers travaux sur les cartes auto-organisatrices [55].

Le taux d'apprentissage $\alpha(t)$ représente le coefficient d'adaptation du vecteur des poids. En général, l'adaptation du vecteur de poids au cours de l'apprentissage est réduite par un coefficient $\alpha(t)$ (où $0 \leq \alpha \leq 1$) qui tend vers 0, après un nombre d'itérations égal à *500 fois* le nombre de neurones sur le réseau pour éviter le sur-apprentissage [60]. En principe, pour une meilleure reproduction de l'espace d'entrée sur la carte, il est préférable que la valeur de ce paramètre soit proche de 1 pour les 1000 premières itérations d'apprentissage. Ensuite, elle commence à décroitre d'une façon linéaire ou exponentielle en fonction de nombre d'itérations.

ii) Phase d'apprentissage

La phase d'apprentissage est la phase de convergence de la couche compétitive vers une topologie similaire à celle de l'espace d'entrée. Pendant cette phase, les vecteurs de poids seront adaptés aux vecteurs de chaque échantillon de la base d'apprentissage qui seront présentés un par un, sur la couche d'entrée, pendant chaque itération. Une itération d'apprentissage passe par deux phases [69] : la phase de compétition et la phase d'adaptation.

La phase de compétition démarre à la réception d'un échantillon $\overrightarrow{X}$ sur la couche d'entrée. L'étape suivante se déroule sur la couche compétitive où chaque neurone calcule la distance Euclidienne entre son vecteur de poids et le vecteur d'entrée de l'échantillon, suivant l'équation 6. Ensuite, l'étape d'élection commence avec une comparaison des distances précédemment calculées par tous les neurones. A la fin de cette étape, un neurone gagnant sera élu suivant l'équation 7. Le neurone vainqueur consiste au neurone qui a calculé la distance minimale. L'identité de ce neurone sera ensuite utilisée au cours de la phase d'adaptation.

Au cours de la phase d'adaptation, chaque neurone doit calculer son taux de voisinage par rapport au neurone gagnant suivant l'équation 8. Ensuite, selon les paramètres d'apprentissage et le taux de voisinage, les vecteurs de poids des neurones concernés par l'adaptation seront mis à jour suivant l'équation 9. Enfin, les paramètres de mise à jour seront également modifiés suivant l'équation 10.

iii) Phase de rappel

La phase de rappel est souvent appelée phase de décision ou phase de classification. En effet, c'est la phase d'utilisation de la carte auto-organisatrice de Kohonen une fois entrainée. Après la fin de la phase d'apprentissage, le réseau est prêt à réaliser des classifications des données de l'espace d'entrée réel. Ces données sont présentées sur la couche d'entrée. Pour chaque vecteur d'entrée, les opérations de la phase de compétition seront exécutées. Donc, les distances Euclidiennes seront calculées et un neurone gagnant sera élu. L'identité de ce neurone sera ensuite utilisée pour identifier la classe du vecteur d'entrée sur la couche de sortie au cours de la phase de décision. Par conséquent, la classification à base de la carte auto-organisatrice de Kohonen se déroule d'une façon itérative.

1.9 Conclusion

Ce chapitre a été consacré aux généralités sur les réseaux de neurones artificiels. Nous avons présenté ce modèle de traitement de données en retournant sur l'historique de son évolution. De plus, nous avons présenté différents modèles neuronaux. Enfin, nous avons détaillé les réseaux de Kohonen et plus précisément la carte auto-organisatrice que nous avons adoptée au cours de ces travaux de recherche.

Ce modèle neuronal est fréquemment implémenté pour des applications relevant des différents domaines : traitement d'images, robotique, compression de données, fusion des données, etc... Chaque application a ses propres caractéristiques et ses exigences. Certaines ont des contraintes de précision, d'autres ont des contraintes temporelles plus ou moins fortes.

Dans la littérature, il existe plusieurs implémentations des réseaux de Kohonen avec des approches logicielles et/ou matérielles. Les implémentations logicielles sont caractérisées par leur flexibilité permettant l'adaptabilité de l'implémentation de la carte *SOM* à différentes applications. Par contre, les limites de ces implémentations résident dans leurs performances sanctionnant leur utilisation pour des applications avec des contraintes temporelles serrées. Par conséquent, pour ces applications, les implémentations matérielles sont favorables, vu leur capacité d'exploiter le parallélisme inhérent des réseaux de Kohonen. Par contre, cette amélioration des performances est au détriment de la flexibilité souvent souhaitée et nécessaire pour certaines applications.

Dans le chapitre suivant, nous nous focalisons sur les principales architectures des réseaux de Kohonen, tant au niveau logiciel qu'au niveau matériel. Les avantages de chacune des approches seront relevées ainsi que leurs principales limitations. Un accent particulier sera mis sur les architectures matérielles, étant donné qu'une utilisation des cartes de Kohonen dans les systèmes embarqués à fortes contraintes de temps est principalement visée par les architectures proposées dans ces travaux de recherche.

Chapitre 2

Etat de l'art sur les implémentations des réseaux de Kohonen

Sommaire

"*Le fait suggère l'idée, l'idée dirige l'expérience, l'expérience juge l'idée.*"

Claude Bernard

2.1 Introduction

Le modèle neuronal, la carte auto-organisatrice ou le réseau de Kohonen, proposé par *Teuvo Kohonen* en 1982 et introduit dans la section précédente, est basé sur une grille d'unités fonctionnelles souvent appelées neurones. Ces unités fonctionnelles ou neurones adoptent le principe des mémoires associatives [47, 54], où chaque neurone sur la carte auto-organisatrice est identifié par sa propre position et détient une information dans son vecteur de poids associé [55]. Les informations stockées dans les vecteurs de poids de neurones d'une carte de Kohonen détiennent un sous-ensemble d'informations choisies de manière aléatoire et propres à l'espace de données d'entrée dont le traitement est envisagé par la carte auto-organisatrice. Cette « extraction » d'information de manière aléatoire de l'espace d'entrée s'effectue lors de la phase d'apprentissage où ces vecteurs d'entrée choisis aléatoirement sont envoyés à tous les neurones pour une comparaison avec leurs poids initiaux. Le résultat de cette « comparaison » est la désignation d'un neurone nommé *neurone gagnant*, se rapprochant par ses propres poids le plus au vecteur d'entrée et une mise à jour des poids des neurones l'avoisinant. Après nombreuses itérations effectuées dans la phase d'apprentissage où ces différents vecteurs d'entrée de l'espace d'entrée sont « projetés » sur la carte auto-organisatrice, l'ensemble de neurones de la carte seront entrainés à l'image de cet espace d'entrée et détiendront dans leurs poids les informations le représentant. Avec cette approche relativement simple, la carte auto-organisatrice proposée par Kohonen présente un outil de traitement d'information capable de s'adapter à n'importe quel espace d'entrée dans la phase d'apprentissage et de manière non supervisée.

D'un point de vue technique, la carte auto-organisatrice ou *Self-Organising Map* (*SOM*) est une représentation algorithmique de l'approche d'apprentissage compétitif. Pendant la phase d'apprentissage, l'organisation de la carte avec cet algorithme se fait automatiquement en fonction des vecteurs d'entrée. Ces derniers représentent l'environnement externe ou l'espace d'entrée sous forme d'un ensemble d'informations dont chacune représente un élément de vecteur. A la fin de l'apprentissage, l'algorithme *SOM* associe un ou plusieurs vecteurs d'entrée à un neurone sur la carte *SOM* selon la similitude entre ce(s) vecteur(s) et son vecteur de poids. Donc, l'idée de cet algorithme revient au principe de « *Clustering* » permettant le regroupement des informations similaires de l'environnement externe (espace d'entrée) de manière non supervisée et auto-adaptative sur la carte de Kohonen finale [48].

De nos jours, l'algorithme *SOM* est l'un des réseaux de neurones à base d'apprentissage non supervisé les plus répandu et utilisé. Il est utilisé dans différentes applications relevant des domaines différents comme traitement d'image, réduction de bruit, correction des données corrompues, compression, etc. Pour certaines applications, notamment dans les applications de classification, l'algorithme *SOM* doit être complété par une phase d'étiquetage permettant d'asso-

cier à un ensemble composé d'un ou de plusieurs neurones de la carte une classe ou étiquette spécifique [89]. Cette phase d'étiquetage supplémentaire est effectuée par l'utilisateur et est en étroite relation avec l'environnement externe dont les entrées doivent être classifiées. Cette version supervisée de l'algorithme *SOM*, appelée *Learning Vector Quantization* (*LVQ*), a également été proposée par *Teuvo Kohonen* [60] et trouve également sa place dans nombreuses applications [90–92]. Ces deux approches, dans leurs versions non supervisée et supervisée *SOM* et *LVQ* respectivement, sont connues dans la littérature sous le nom des *réseaux de Kohonen*.

Étant donné le grand intérêt de l'algorithme de Kohonen dans la classification des données multidimensionnelles, l'aspect de son implantation sur des supports physiques divers et variés est une problématique de longue date qui est à l'origine de nombreux travaux de recherche [83, 93–123]. L'aspect implantation de l'algorithme de Kohonen est étroitement lié à l'application et a ses caractéristiques et contraintes comme précision, temps d'exécution, contraintes temps réel, etc. Dans la littérature, on trouve des exemples d'implémentations de l'algorithme de Kohonen basés sur des approches logicielles [83,93–105], des approches matérielles [106–121] ou des approches mixtes [122, 123]. Chaque type d'implémentation a ses propres avantages et inconvénients, et est comme évoqué précédemment étroitement lié à l'application pour laquelle il est destiné.

Dans la suite de ce chapitre, nous nous intéressons en particulier à la version non supervisée des *réseaux de Kohonen*, l'algorithme *SOM*. Nous présenterons l'état de l'art des implémentations de cet algorithme sur des supports dits logiciels, matériels et mixtes, en s'intéressant en particulier sur des architectures matérielles de l'algorithme de Kohonen. De surcroit, nous présenteront également une discussion critique des architectures matérielles que l'on trouve dans la littérature, en vue d'identifier les verrous à lever pour apporter à ces architectures matérielles plus de flexibilité et les rendre extensibles et le plus indépendantes possibles de l'environnement externe ou de l'espace d'entrée et ainsi des applications de manière générale.

2.2 Approches logicielles de l'implémentation de la carte auto-organisatrice

Les approches logicielles permettant l'implémentation de la carte auto-organisatrice consistent en utilisation de supports physiques à base de processeurs à usage général, de traitement de signal ou spécifique comme processeurs graphiques. L'utilisation de ces architectures à base de processeurs sous-entend que toute la flexibilité des applications et systèmes les utilisant réside dans la couche logicielle embarquée.

La mise en œuvre des implémentations logicielles des *cartes de Kohonen* est relativement simple [73, 93, 94, 124]. L'algorithme de Kohonen est embarqué sur un des supports logiciels cités précédemment en le traduisant sous forme d'un code dans un langage de programmation comme *C/C++*, *JAVA*, *Python* ou autre. Le code décrivant l'algorithme de Kohonen dans un langage de programmation sera exécuté de manière séquentielle sur le support logiciel choisi et sa vitesse d'exécution sera principalement dépendante des caractéristiques de ce dernier et des

outils de compilation utilisés pour sa traduction en mode machine. L'avantage principal des solutions logicielles est leur degré de flexibilité élevé permettant des les adapter plus facilement à des scénarios d'exécution différents et à une large gamme d'applications relavant des différents domaines. Cette flexibilité est au détriment des performances limitant l'utilisation des approches logicielles à des applications n'ayant pas de contraintes de temps fortes à gérer. Dans la suite de cette section, nous décrivons davantage la flexibilité des approches logicielles et donnons quelques exemples d'accélération permettant d'augmenter leurs performances tout en gardant la flexibilité initiale. Dans [124], *Kohonen* et *al.* ont proposé une implémentation logicielle de la carte auto-organisatrice. Cette implémentation est proposée comme un package open source permettant l'intégration rapide de la carte *SOM* dans une application développée en langage *C/C++*. Le code source proposé peut être facilement paramétré pour l'adapter à n'importe quelle application.

2.2.1 Flexibilité des implémentations logicielles

Les implémentations logicielles de la *carte auto-organisatrice* sont caractérisées par leur flexibilité. En effet, la flexibilité de ces approches d'implémentation est l'un de leurs atouts principaux. Dans les implémentations logicielles, la structure de la couche compétitive de la carte auto-organisatrice ainsi que le nombre d'éléments des vecteurs utilisés sur la couche d'entrée peuvent généralement être modifiés au cours de l'exécution [106]. De plus, la topologie de distribution ainsi que la topologie de voisinage peuvent également être modifiables dans ce type d'approche, sans avoir besoin de modifications majeures. Ce type de flexibilité est souvent obtenu via des paramètres statiques ou dynamiques, pouvant être choisis soit avant la phase de compilation ou à l'exécution.

D'autre part, les performances des supports physiques utilisés ainsi que la propriété de flexibilité des approches logicielles sont à l'origine des nouvelles variantes de la carte auto-organisatrice. Ces nouvelles approches de l'algorithme *SOM* exploitent non seulement la flexibilité de la structure *SOM* mais également la flexibilité des topologies de la couche compétitive et sont à l'origine d'un réseau de neurones ayant une structure variable au cours de la phase d'apprentissage. La figure 2.1 présente trois exemples d'architecture logicielle croissante résultant de cette flexibilité des approches logicielles.

Dans [100], *Fritzke* a proposé une architecture à base de *SOM* nommée *Growing Grid SOM*. Cette architecture distribuée avec une topologie *2D maillée*, s'incrémente en termes de nombre de lignes et de colonnes au cours de l'apprentissage, permettant ainsi d'adapter le nombre de neurones de la carte *SOM* en fonction des besoins applicatifs et critères par exemple de précision propres à l'application donnée. Dans [102], *Sasamura et al.* ont proposé une architecture similaire nommée architecture incrémentale. L'architecture proposée a une topologie *irrégulière*, dans laquelle le nombre de neurones sur la couche compétitive s'incrémente au cours de la phase d'apprentissage. Donc, un neurone pourra être ajouté sur la carte après un certain nombre d'itérations d'apprentissage. De plus, dans cette architecture, la topologie de voisinage pourra également

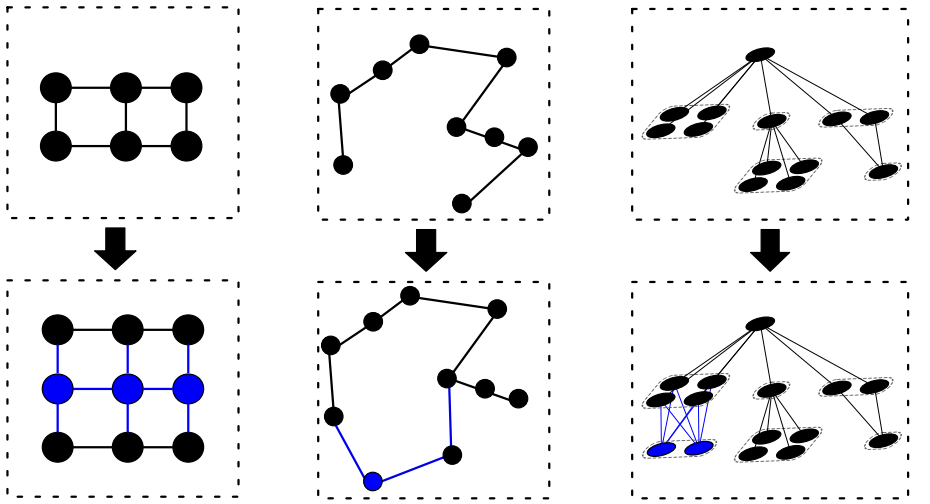


FIGURE *2.1 – Architecture de la carte auto-organisatrice à structure dynamiquement variable*

changer en ajoutant ou en éliminant des interconnexions entre les neurones existants de la carte avant l'opération d'incrémentation. Une autre approche similaire a été proposée par *Rauber et al.* dans [101]. Cette architecture est basée sur une topologie hiérarchique distribuée. Dans cette architecture, un ensemble de neurones sur le niveau hiérarchique le plus bas pourra être rajouté à la carte au cours de l'apprentissage, si besoin.

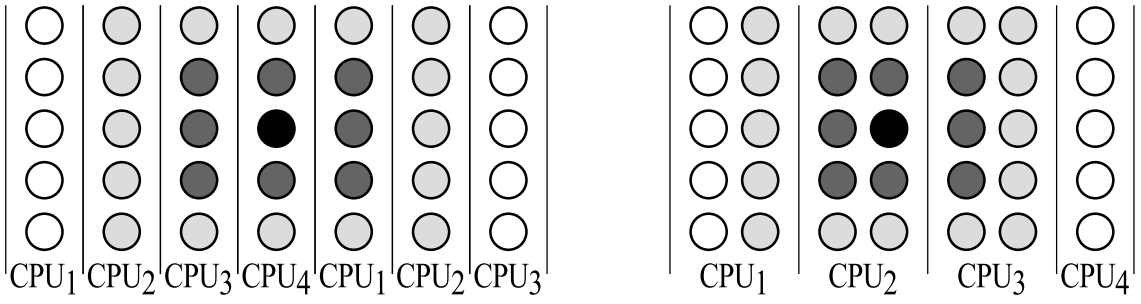
En contrepartie, pour des applications ayant des contraintes temporelles fortes, les implémentations de la carte auto-organisatrice à base des approches logicielles sont inadaptées voire déconseillées, principalement à cause des performances en termes de temps d'exécution obtenues étant modestes voire insuffisantes pour une exécution temps réel. En effet, l'exécution séquentielle de l'algorithme de Kohonen ne permet pas de profiter pleinement du parallélisme inhérent de l'algorithme, qui distribué convenablement sur une architecture à grain fin pourra offrir des performances nettement plus élevées que celles obtenues sur des supports logiciels classiques [95]. D'ailleurs, dans les architectures logicielles, des travaux d'accélération de l'algorithme de Kohonen visant à améliorer ces performances temporelles seront présentés dans la section suivante.

2.2.2 Approches d'accélération des implémentations logicielles

Dans la littérature, nous trouvons des exemples de travaux de recherche dont l'objectif principal est d'améliorer les performances temporelles des implémentations logicielles de la carte auto-organisatrice de Kohonen. Une première approche que nous trouvons consiste à simplifier le traitement de l'algorithme de Kohonen permettant de réduire le nombre et la durée d'exécution d'instructions nécessaires à la réalisation de cet algorithme [103, 104]. Les simplifications proposées consistent à remplacer les équations de calcul de la distance *Euclidienne* et de calcul du taux de voisinage respectivement par le calcul de la distance *Manhatan* et une simple opération de division de la différence entre le vecteur d'entrée et le vecteur de poids en fonction de

l'ordre de voisinage par rapport au neurone gagnant. Ces simplifications permettent d'accélérer le traitement sans influencer les résultats de l'algorithme de Kohonen, au cours de la phase d'apprentissage non supervisé ainsi que la phase de rappel. Par contre, le parallélisme de l'algorithme de Kohonen n'est pas complètement exploité via ces approches.

Entres autres, les supports à base des architectures multi-processeurs, permettant une exécution parallèle des instructions n'ayant pas de dépendance de flux de données, se prêtent bien au parallélisme inhérent de l'algorithme de Kohonen et peuvent être utilisés pour son accélération. Dans la littérature, on trouve des travaux de recherche proposant des solutions d'accélération sur des supports multi-processeurs (*multi-CPU* (1)), multi processeurs graphiques (*multi-GPU* (2)) ou sur des supports combinant les deux précédents. L'idée consiste à répartir l'exécution d'instructions des neurones sur différents processeurs. Par conséquent, la concurrence matérielle sera distribuée ainsi que le temps d'attente de libération du processeur. Dans [83], *Kolinummi et al.* ont proposé une implémentation logicielle parallèle de la carte auto-organisatrice. L'exécution de cette carte est effectuée sur une architecture *Multi-Processor System-on-Chip* (*MP-SoC*). L'affectation des neurones d'une carte *SOM* 2D maillée sur les processeurs de l'architecture *MP-SoC* est effectuée par colonne. Ainsi, chaque processeur se charge de l'exécution de l'ensemble de neurones de la colonne qui lui a été attribuée, comme présenté sur la figure 2.2(a). Ce mode de répartition de neurones d'une carte *SOM* 2D maillée est appelé *colonnes adjacentes sur processeurs adjacents*. D'autre part, *Hamalainen et al.* ont proposé dans [95] une nouvelle approche de répartition de neurones d'une carte *SOM* 2D maillée sur un support multi-processeurs. La répartition des neurones est toujours effectuée par colonnes, en mode *plusieurs colonnes adjacentes sur un même processeur*. Ce mode de répartition est illustré sur la figure 2.2(b), où les neurones appartenant à des colonnes adjacentes sont affectés et exécutés sur un même processeur. Ces approches d'accélération de l'algorithme de Kohonen [83, 95] ont permis d'obtenir des meilleures performances d'exécution des étapes clés de l'algorithme de Kohonen, notamment du calcul de la distance et de l'adaptation. Dans les travaux présentés, une accélération de 4 fois par rapport à une architecture simple processeur a été observée.

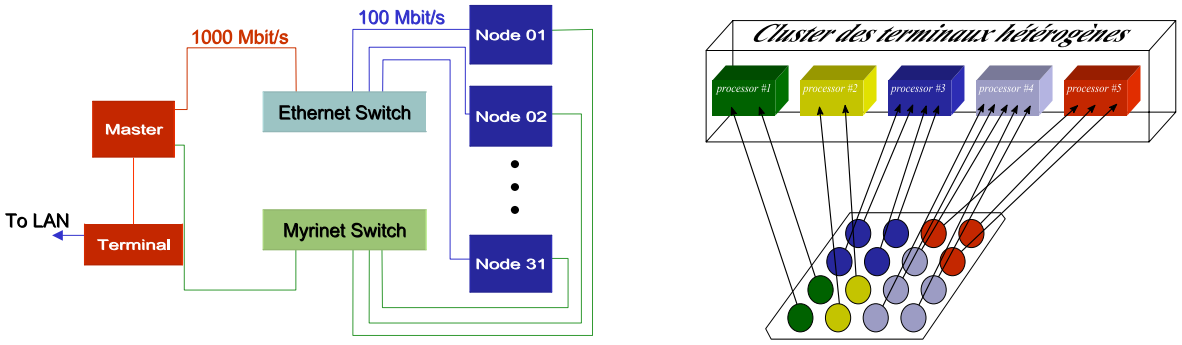


(a) Répartition: neurones adjacents sur CPU adjacents (b) Répartition: neurones adjacents sur même CPU

FIGURE *2.2 – Répartitions des neurones d'une carte SOM 2D maillée sur un support multi-processeurs*

(1). *Central Processing Unit* (*CPU*)
(2). *Graphics Processing Unit* (*GPU*)

Une autre approche de parallélisation de l'algorithme de Kohonen a été proposée par *Seiffert* dans [105]. Cette approche repose sur l'exécution parallèle de la carte de Kohonen sur des *clusters* de processeurs connectés en réseau (*LAN clusters*, voir figure 2.3(a)). L'idée consiste à partitionner la carte *SOM* en plusieurs parties et d'affecter chaque partie comprenant un ensemble de neurones sur des clusters de processeurs *multi-cœur* synchronisés entre eux via une communication Ethernet. Pour éviter les retards de communication conséquents pendant la phase de compétition et de décision, le voisinage topologique entre les neurones de la carte *SOM* est pris en compte. Par conséquent, les neurones topologiquement proches et appartenant à la même classe sont exécutés sur le même cluster de processeur comme présenté sur la figure 2.3(b).



(a) Architecture du Cluster Réseau

(b) Répartition des neurones sur les processeurs

FIGURE 2.3 – *Approche d'implémentation logicielle parallèle sur un cluster de processeurs connectés en réseau LAN*

D'un autre côté, des implémentations logicielles sur des supports *multi-GPU* ont souvent été proposées pour des applications de traitement d'image. En général, l'approche d'exécution parallèle utilisée pour les supports multiprocesseurs est également employée dans ce cas [96, 97], à la différence près que les données à traiter peuvent également être réparties permettant ainsi de traiter plusieurs échantillons d'entrée en parallèle [99]. Pour ces supports multi-GPU, des outils de développement ainsi que des bibliothèques prédéfinies sont disponibles pour faciliter la répartition du calcul à effectuer sur différents nœuds de calcul. A titre d'exemple, les calculs à exécuter sur les plateformes multi-GPU peuvent être configurés via des interfaces de programmation comme *Compute Unified Device Architecture (CUDA)* ou *Open Graphics Library (Open-GL)*, qui permettent également de cibler des architectures hétérogènes et mixtes combinant les *CPU aux GPU* [98].

Malgré les approches d'optimisation du temps d'exécution des implémentations logicielles de l'algorithme de Kohonen, ces dernières ne permettent pas de répondre à des contraintes temporelles fortes souvent présentes dans les systèmes embarqués et dans des applications temps réel. De surcroît, les solutions d'accélération proposées sont relativement couteuses en termes de ressources matérielles utilisées ainsi qu'en termes de consommation d'énergie. Par conséquent, pour des applications embarquées et à des contraintes de temps fortes, les implémentations matérielles peuvent être privilégiées aux solutions dites logicielles [107]. Dans la section suivante, nous don-

nons une synthèse des architectures matérielles de l'algorithme de Kohonen que nous trouvons dans la littérature, avec leurs principaux avantages et inconvénients.

2.3 Approche matérielle de l'implémentation de la carte auto-organisatrice

L'algorithme de Kohonen comprend un certain nombre de fonctions dont l'exécution repose sur des données indépendantes. Par conséquent, ces fonctions peuvent être exécutées simultanément sur des modules de calcul distribués fonctionnant en parallèle. Comme nous l'avons vu dans la section précédente, ce parallélisme est souvent exploité par des approches de programmation parallèle à base des supports multiprocesseurs. D'un autre côté, des implémentations matérielles où pour chaque fonction de l'algorithme de Kohonen un nœud de calcul adapté et conçu spécifiquement est utilisé, sont également proposées dans la littérature. Ces approches matérielles, en plus des meilleures performances temporelles par rapport aux approches logicielles, permettent non seulement d'optimiser le coût de réalisation mais également de réduire la consommation d'énergie totale augmentant ainsi l'autonomie énergétique des systèmes les embarquant [107].

La carte auto-organisatrice de Kohonen est souvent utilisée pour des traitements temps réel caractérisés par des contraintes temporelles fortes. La granularité fine des approches matérielles permettant de proposer à chaque fonction à réaliser un circuit adapté, met les implémentations matérielles des réseaux de Kohonen au premier plan en les présentant comme solution pour répondre à ces besoins de calcul rapide. Les implémentations matérielles des cartes de Kohonen proposées dans la littérature ont largement démontré leur supériorité par rapport aux implémentations logicielles [106]. Pour comparer les performances temporelles de deux implémentations différentes de carte auto-organisatrice, on utilise souvent deux métriques : *Million Connection Per Second* (*MCPS*) et *Million Connection Updates Per Second* (*MCUPS*). Ces métriques représentent respectivement les performances de la phase de rappel (équation 2.1) et les performances de la phase d'apprentissage (équation 2.2) :

$$MCPS = \frac{D \times N_{neurones}}{T_{IR}} \times f \cdot 10^{-6} \tag{2.1}$$

$$MCPS = \frac{D \times N_{neurones}}{T_{IA}} \times f \cdot 10^{-6} \tag{2.2}$$

avec D étant le nombre d'éléments sur la couche d'entrée ; $N_{neurones}$ le nombre de neurones sur la couche compétitive ; T_{IR} le nombre de cycles d'horloge pour l'exécution d'une itération au cours de la phase de rappel ; T_{IA} le nombre de cycles d'horloge pour l'exécution d'une itération au cours de la phase d'apprentissage ; et f la fréquence de fonctionnement du support d'exécution.

La structure de la carte auto-organisatrice de Kohonen sur une implémentation matérielle est souvent prédéfinie en fonction de la dimension de la couche d'entrée D et de la taille de la couche compétitive $N_{neurones}$. Par conséquent, l'amélioration des performances d'une implémentation

matérielle d'une structure prédéfinie consiste, d'une part, à réduire le nombre de cycles nécessaires pour l'exécution des opérations de l'algorithme de Kohonen. D'autre part, on cherche à augmenter la fréquence de fonctionnement du support physique en optimisant au maximum les circuits proposés pour réaliser les fonctions spécifiques de l'algorithme de Kohonen.

2.3.1 Adaptations de l'algorithme de Kohonen dédiées aux implémentations matérielles

Plusieurs opérations de l'algorithme de Kohonen comme le calcul de la distance Euclidienne, le calcul du taux de voisinage et l'adaptation des vecteurs de poids, nécessitent l'utilisation des opérateurs arithmétiques complexes pour leur mise en œuvre. Ces opérateurs arithmétiques consomment beaucoup de ressources matérielles d'une part, et peuvent influencer les performances temporelles du système complet principalement à cause de leur complexité réduisant la fréquence de fonctionnement maximale du système, d'autre part. Pour éviter ces goulots d'étranglement du point de vue performances, ces opérations dites complexes de l'algorithme de Kohonen peuvent être simplifiées en vue des implémentations matérielles de la carte auto-organisatrice de Kohonen. Quelques adaptations de l'algorithme de Kohonen sont présentées par *Teuvo Kohonen* dans son livre *Self-Organizing Maps* [55].

La mise en œuvre du calcul de la distance *Euclidienne* présentée par l'équation 2.3, nécessite des opérations de multiplication ou de mises au carré, d'addition ou d'accumulation ainsi que l'opération de racine carrée [109, 111]. Pour simplifier cette mise en œuvre, une solution adoptée dans nombreux travaux [110, 112], consistent à remplacer la distance *Euclidienne* de la *norme L2* (D_{L2}) par la distance *Manhattan* de la *norme L1* (D_{L1}) exprimée par l'équation 2.4. Cela permet d'éviter l'utilisation des opérateurs de multiplication pour le calcul du carré de la différence entre les éléments des vecteurs de poids et d'entrée, d'une part ; et d'éliminer l'opération de la racine carrée d'autre part. Par conséquent, les opérateurs arithmétiques utilisés pour le calcul de cette distance seront uniquement des opérateurs d'addition et de soustraction.

$$D_{L2} = \left\| \overrightarrow{X} - \overrightarrow{W} \right\| = \sqrt{(\xi_1 - \mu_{i,1})^2 + (\xi_2 - \mu_{i,2})^2 + \cdots + (\xi_D - \mu_{i,D})^2} \tag{2.3}$$

$$D_{L1} = \left\| \overrightarrow{X} - \overrightarrow{W} \right\| = \left|\xi_1 - \mu_{i,1}\right| + \left|\xi_2 - \mu_{i,2}\right| + \cdots + \left|\xi_D - \mu_{i,D}\right| \tag{2.4}$$

Bien que cette simplification permette de réduire le coût d'implantation en termes de ressources, elle peut agir sur les résultats d'élection du neurone gagnant pendant la phase de compétition. Tel qu'il est présenté sur l'exemple de la figure 2.4, on peut avoir deux distances différentes $D(a,b)$ et $D(a,c)$ avec $D(a,b) > D(a,c)$ qui donneront des résultats différents voir contradictoires du point de vue de distances L_1 et L_2. Dans cet exemple, au niveau de la distance *Manhattan* D_{L1}, on remarque que les deux distances sont égales, tandis qu'au niveau de la distance *Euclidienne* $\mathrm{D}_{\mathrm{L}_2}$ les deux distances ne sont pas identiques, étant le résultat attendu vu la condition initiale $D(a,b) > D(a,c)$. De plus, sur la figure 2.4 on peut voir la représentation graphique de

l'égalité des distances de différente norme : pour la distance *Euclidienne* (L_2) et sa valeur mise au carré, il s'agit d'un rayon dessinant un cercle autour du point de départ tandis qu'au niveau de la distance *Manhattan* (L_1) il s'agit d'un carré dont le croisement des ces deux diagonales représente le point de départ. Dans le cas de la distance *Mahnattan*, on peut constater que tous les points ayant la même distance ne sont pas uniformément distribués autour du point de départ. Cela peut influencer la phase d'apprentissage de la carte de Kohonen ainsi que la précision pendant la phase de rappel [72].

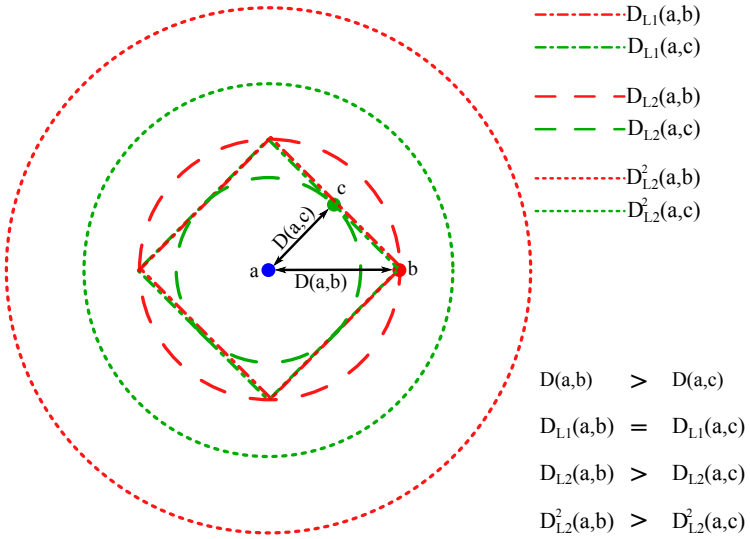


FIGURE *2.4 – Comparaison entre les différentes équations de calcul de la distance* (D_{L1}, D_{L2} *et* D^2_{L1})

Une autre solution préférée dans d'autres travaux de recherche est de maintenir la qualité d'élection du neurone gagnant obtenue avec le calcul de la distance *Euclidienne* en n'utilisant que le carré de cette distance. De cette façon, l'implantation de l'opération de la racine carrée est évitée [81, 106, 113]. Cette approche repose sur l'idée, présentée graphiquement sur la figure 2.4 : si $D_{L2}(a,b) > D_{L2}(a,c)$ alors $D^2_{L2}(a,b) > D^2_{L2}(a,c)$. Avec cette approche de simplification, la précision des résultats obtenus avec la distance *Euclidienne* est conservée. D'un autre côté, la taille des registres nécessaires pour l'enregistrement et la sauvegarde des distances Euclidiennes est multipliée par deux. Dans [81], *Dlugosz et al.* ont proposé une architecture configurable permettant d'utiliser les deux normes *L1* et *L2*. Pour enregistrer les résultats de deux normes dans un registre de taille fixe, les auteurs ont choisi d'utiliser l'opération de la racine carrée dans le calcul de la distance Euclidienne. Dans [114], *Kurdthongmee* a proposé une nouvelle architecture de la carte de Kohonen à base d'une *mémoire RAM* indexée par la distance *Euclidienne* calculée. Dans cette approche, pour un vecteur d'entrée donné à un instant donné (une itération), l'identité du neurone gagnant représente le contenu de la case mémoire dont l'adresse ou l'index correspond à la distance Euclidienne calculée par le neurone entre son vecteur de poids et le vecteur d'entrée. Pour réduire la taille de la mémoire sans perdre au niveau de la précision des résultats obtenus, l'auteur a utilisé la distance *Euclidienne* calculée selon l'équation originale en utilisant l'opérateur de racine carrée. Comme indiqué précédemment, cette distance donne une meilleure

précision par rapport à la distance *Manhattan* d'une part, et en plus elle est codée sur la moitié des bits nécessaires pour représenter la distance *Euclidienne* au carré, d'autre part.

Dans l'article [108], *Talaska et al.* ont proposé un circuit analogique programmable de calcul de distance. Ce circuit de calcul nommé *Distance Calculation Circuit* (*DCC*) peut être configuré pour calculer soit la distance *L2* ou *L1* selon les besoins de précision de l'application. Ce circuit proposé, peu gourmand en termes de ressources, permet de calculer la distance *Euclidienne* entre deux vecteurs de trois éléments en *un cycle d'horloge* en fonctionnant à une fréquence de *10 MHz*.

D'autre part, le calcul du taux voisinage $h(t)$ présenté par l'équation 2.5 est également d'une grande complexité pour une implémentation matérielle. Ce calcul demande l'implantation des opérateurs de multiplication, de division et exponentiel ainsi que les opérateurs d'addition et de soustraction. Dans la littérature, on trouve plusieurs travaux de recherche dans lesquels des solutions se rapprochant de la fonction *gaussienne* ont été proposées pour simplifier l'équation 2.5 [72, 115]. Une autre approche de simplification de l'équation 2.5 trouvée dans plusieurs travaux traitant de l'implémentation matérielle des cartes de Kohonen est de simplifier cette équation en utilisant *une simple opération de décalage* [2, 125]. Comme illustré sur la figure 2.5, cette opération de décalage permet de réaliser une fonction de division par 2^S, avec *S* étant l'ordre de voisinage du neurone. En prenant en compte le cercle de voisinage sur la distribution topologique des neurones ayant comme centre le neurone gagnant, et ayant comme rayon l'ordre de voisinage du neurone le plus éloigné à adapter, la simplification proposée sera utilisée si cet ordre est inférieur au rayon de voisinage, sinon le taux de voisinage sera égal à *0*, et donc le vecteur de poids ne sera pas affecté par l'adaptation.

$$h_{C,i}(t) = \exp(-\frac{\left\| \overrightarrow{r_c} - \overrightarrow{r_i} \right\|}{2\sigma^2(t)}) \tag{2.5}$$

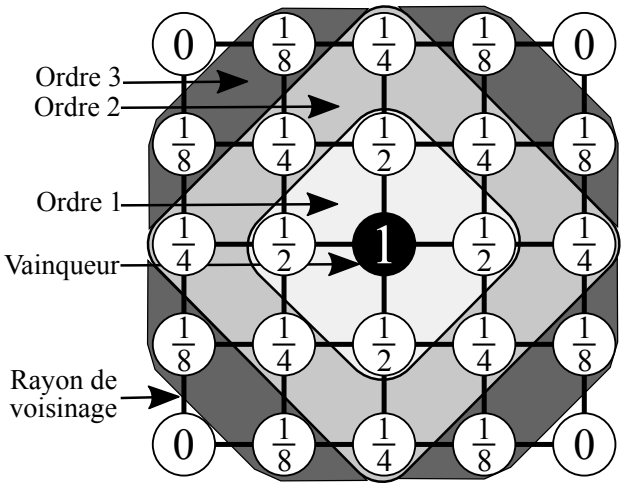


FIGURE *2.5 – Simplification de l'équation de voisinage [2]*

Dans [126], *Ruping et al.* ont analysé l'influence de la simplification de l'équation 2.5 par

une simple opération de décalage. Les simulations ont montré que les résultats obtenus par l'algorithme de Kohonen à base d'une fonction de voisinage *gaussienne* (l'équation 2.5) sont comparables à ceux obtenus avec la version développée à base d'une simple fonction de division par décalage. D'un autre côté, selon les auteurs, le seul inconvénient, en cas d'utilisation de cette simplification, est le nombre d'itération d'apprentissage qui sera plus élevé par rapport à l'algorithme de Kohonen classique. Ce nombre d'itérations supplémentaire permet d'arriver au même stade d'entrainement d'une carte de Kohonen utilisant l'algorithme de Kohonen sans simplifications.

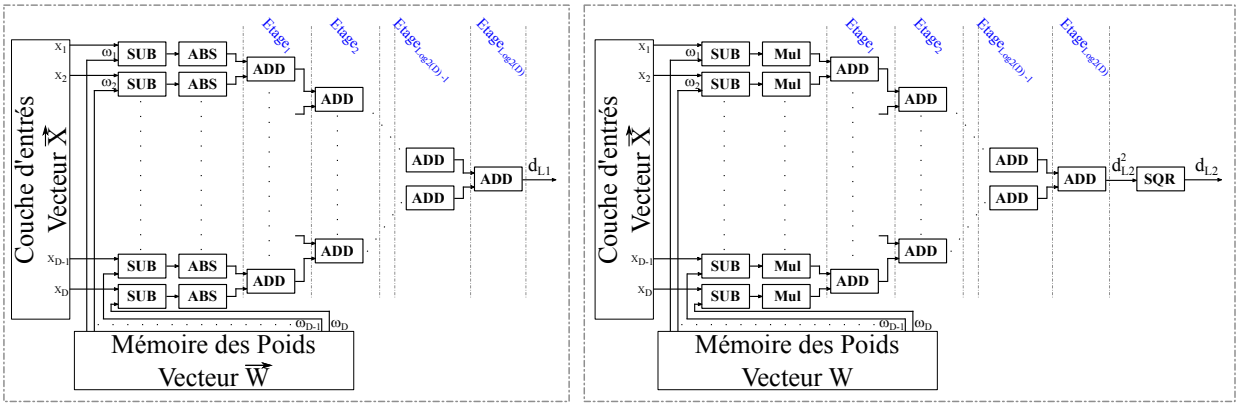
Dans le cadre de simplification de la fonction de calcul du taux de voisinage, *Tamukoh et al.* ont proposé une nouvelle approche d'adaptation des vecteurs de poids, dans [118]. Cette approche appelée *Rough-Winner-Take-All* (*RWTA*) ne repose pas sur la fonction de voisinage $h(t)$ présentée par l'équation 2.5. L'idée est de remplacer le voisinage de la distribution topologique des neurones sur la couche compétitive de la carte de Kohonen $\left\| \overrightarrow{r_c} - \overrightarrow{r_i} \right\|$ par un voisinage calculé en termes de proximité des vecteurs de poids sur l'espace réel. En effet, pour chaque stimulus, les distance calculées seront décalées à droite de *R(t)* bits, avec *R(t)* étant le rayon de voisinage. Ensuite, seulement les neurones ayant les distances décalées minimales seront affectés par l'adaptation de leurs vecteurs de poids avec un taux d'apprentissage exprimé par un décalage à droite de *R(t)* bits de la différence entre le vecteur d'entrée et le vecteur de poids. Enfin, on procède à la réduction du rayon de voisinage *R(t)*. De cette façon, l'adaptation par taux de voisinage sera effectuée au cours de l'apprentissage itératif avec la réduction du paramètre *R(t)* après un certain nombre d'itération d'apprentissage, ce qui engendre à la fois la minimisation du nombre de neurones affectés par l'adaptation et l'augmentation du taux d'apprentissage de ces neurones. Cette approche permet de réduire le nombre d'itération d'apprentissage afin de converger rapidement vers la projection de l'espace d'entrée.

A base des simplifications précédemment présentées, plusieurs architectures matérielles de la carte auto-organisatrice ont été proposées dans la littérature [2,84,111,125,127]. Ces architectures ont souvent été proposées dans un cadre applicatif caractérisé par des performances élevées exigées ou une consommation d'énergie réduite. Plusieurs approches d'implémentation matérielles ont été proposées selon les besoins de l'application. Parmi les architectures matérielles les plus courantes, on trouve trois types d'architecture : architecture massivement parallèle, architecture sérielle parallèle et architecture séquentielle.

2.3.2 Architecture massivement parallèle de la carte SOM

L'architecture parallèle de la carte auto-organisatrice de Kohonen permet de distribuer le calcul de l'algorithme de Kohonen sur plusieurs modules ou éléments de calcul. Chaque module représente en général un neurone sur la couche compétitive de la carte *SOM*. Ces modules de calcul ou neurones effectuent simultanément le calcul de la distance L_2 et/ou L_1 entre les vecteurs d'entrée et leurs vecteurs de poids associés de manière générale en utilisant des équations simplifiées, comme présenté dans la section précédente. Dans ce cas, on parle d'une architecture

de type *Single Instruction Multiple Data* (*SIMD*), où une instruction unique (le calcul de la distance L_2 et/ou L_1) est exécutée sur des données multiples (un même vecteur d'entrée pour tous les modules et des vecteurs de poids différents pour chaque module). Dans cette architecture, les éléments du vecteur d'entrée présentés sur la couche d'entrée sont traités en parallèle sur des modules distribués au niveau de chaque neurone. Cette architecture est souvent appelée *architecture massivement parallèle*.



(a) Architecture Parallèle pour le calcul de la distance Manhattan

(b) Architecture Parallèle pour le calcul de la distance Euclidienne

FIGURE 2.6 – *Architectures matérielles parallèles pour le calcul des distances : (a) Norme* L_1*, (b) Norme* L_2

Comme illustré sur la figure 2.6, le calcul de la distance suivant les deux normes L_1 ou L_2 s'effectue en parallèle sur les différents éléments des vecteurs d'entrée $\overrightarrow{X}$ et de poids $\overrightarrow{W}$. Pour chaque paire d'éléments (x_i, w_i), où x_i et w_i sont des éléments des vecteurs $\overrightarrow{X}$ et $\overrightarrow{W}$ ($0 < i < D$, D étant la dimension des vecteurs $\overrightarrow{X}$ et $\overrightarrow{W}$), on calcule la différence en utilisant un opérateur de soustraction. Ensuite, le carré sur la norme L_2 ou la valeur absolue sur la norme L_1 pour chaque paire sera calculé respectivement en utilisant une opération de multiplication ou une opération d'addition [111]. Finalement, la distance L_1 ou L_2 sera calculée par l'accumulation des différences de chaque paire déjà calculées. La fonction de l'accumulation s'effectue simultanément en utilisant un ensemble d'opérateurs d'addition distribués sur plusieurs étages. Le nombre d'étage d'additionneur est calculé en fonction de la dimension de la couche d'entrée suivant l'équation 2.6. Les additionneurs appartenant au même étage fonctionnent en parallèle.

$$E_A = Log_2(D) \tag{2.6}$$

avec E_A étant le nombre d'étages d'addition et *D* étant la dimension des vecteurs $\overrightarrow{X}$ et $\overrightarrow{W}$.

Avec ce type d'architecture, toutes les distances L_2 ou L_1 seront calculées en parallèle. La comparaison de ces distances au cours de la phase de compétition, afin d'élire le neurone gagnant, s'effectue ainsi en parallèle. Cette opération est assurée par un comparateur global comme présenté sur la figure 2.7. Ce comparateur est composé d'un ensemble de comparateurs élémentaires à deux entrées distribués sur des étages successifs. Le nombre d'étages est calculé en fonction du nombre de modules sur la couche compétitive avec l'équation 2.7 :

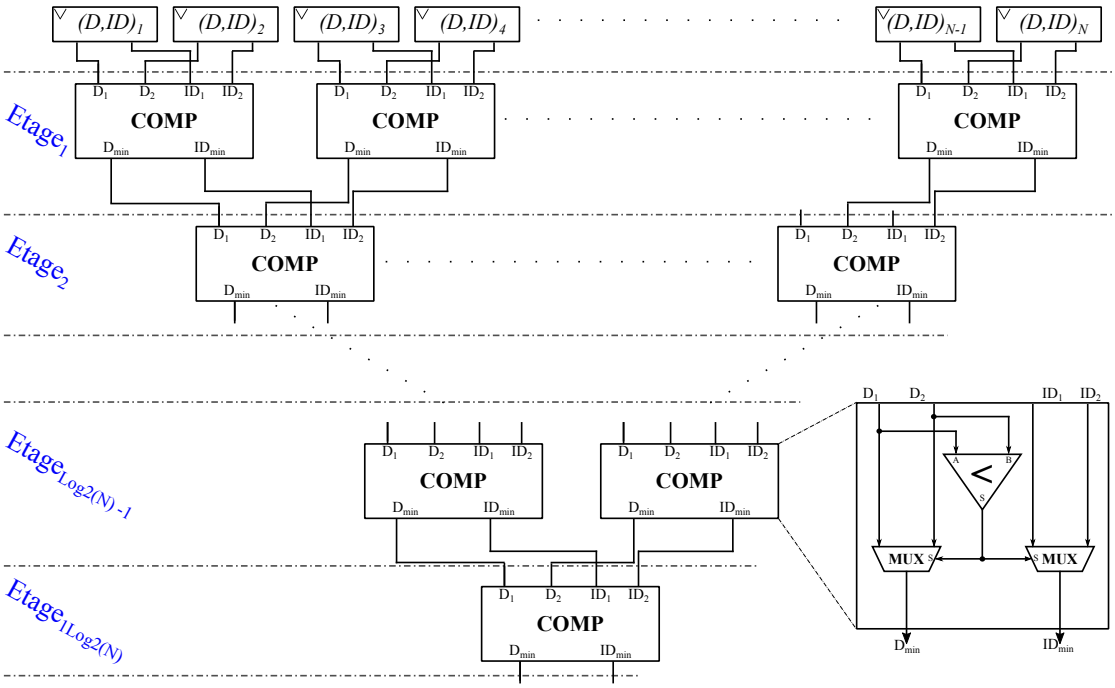


FIGURE 2.7 – *Architecture parallèle du comparateur global*

$$E_C = Log_2(N_{neurones}) \tag{2.7}$$

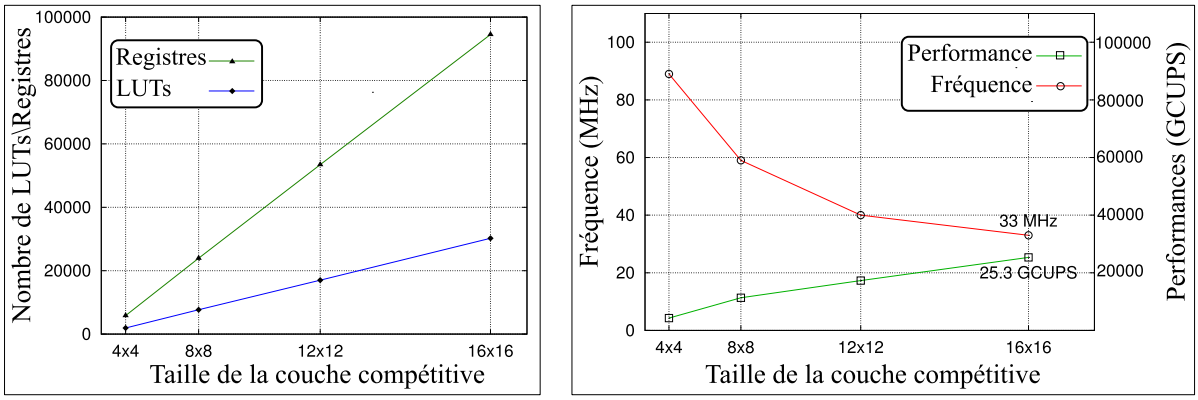
avec E_C étant le nombre d'étages de comparaison et $N_{neurones}$ étant le nombre de neurones sur la couche compétitive. Par conséquent, dans ce type d'architecture, les opérations de comparaison sur un étage *i* seront effectuées simultanément.

L'architecture matérielle massivement parallèle permet d'exploiter le parallélisme de l'algorithme de Kohonen d'une façon plus optimale. Dans [125], *Ben Khalifa et al.* ont proposé une architecture massivement parallèle permettant d'exécuter chaque opération de l'algorithme en parallèle en un cycle d'horloge. Ces résultats sont obtenus sur une carte *SOM* composée de *25 neurones* et traitant des vecteurs d'entrée d'une taille de *23*. Dans cette architecture, implantée sur une carte *FPGA* de type *RC100-PP* et tournant à une fréquence d'horloge de *25 MHz*, une itération de rappel peut être exécutée en 1.37μs. Par conséquent, les performances de cette architecture au cours de la phase de rappel sont de *419.7 MCPS*. Dans cette architecture, on remarque que sa fréquence de fonctionnement maximale est relativement faible. Cela revient au modèle du support *FPGA* utilisé, d'une part. D'autre part, l'exécution parallèle du calcul de la distance *Euclidienne* peut avoir une grande influence sur le chemin critique du circuit final et ainsi sur la fréquence maximale de fonctionnement. En effet, le calcul de la distance Euclidienne en *un seul cycle d'horloge* passe par un opérateur de soustraction, un opérateur de multiplication ainsi que *5* opérateurs d'additions successives. Par conséquent, un cycle d'horloge doit être supérieur à la somme des délais nécessaires pour effectuer toutes ces opérations. De surcroît, si on augmente la dimension du vecteur de poids/d'entrée avec cette approche la fréquence maximale de fonctionnement diminuera à cause de l'ajout des étages d'additionneurs supplémentaires.

Pour améliorer les performances des architectures massivement parallèles, plusieurs travaux dans la littérature ont proposé une approche consistant à synchroniser l'exécution des différentes opérations en les découpant sur plusieurs cycles d'horloge [117]. Par conséquent, chaque opération sera exécutée sur plusieurs cycles d'horloge dont le nombre dépend de la dimension du vecteur des poids pour le calcul de la distance (L_1 ou L_2) et du nombre de neurones sur la couche compétitive pour la recherche du neurone gagnant. Dans le travail présenté en [111], *Ramirez-Agundis et al.* ont proposé une architecture matérielle massivement parallèle d'une carte auto-organisatrice composée de *256 neurones* avec un vecteur de poids d'une dimension de *16*. Cette architecture utilisant la distance *L1* permet d'améliorer la fréquence de fonctionnement de la carte *SOM* implantée, qui atteint *100 MHz*, avec une carte *FPGA Virtex XC2V6000-6*. Dans cette architecture, une itération de rappel nécessite *26 cycles d'horloge* tandis qu'une itération d'apprentissage prend *45 cycles d'horloge*. Par conséquent, les performances obtenues avec cette architecture sont de *15 754 MCPS* pendant la phase de rappel et de *9 102 MCUPS* pendant la phase d'apprentissage. On remarque que le nombre de cycle d'horloge nécessaires pour effectuer les étapes d'apprentissage et de rappel augmente avec cette approche par rapport à une approche massivement parallèle à faible latence présentée auparavant. Concernant la fréquence de fonctionnement maximale de cette architecture, elle augmente également par rapport à une architecture massivement parallèle à faible latence car elle dépend du chemin critique qui dans ce cas traverse moins d'opérateurs arithmétiques. Cette architecture est également plus robuste aux changements de la structure de la carte *SOM*, en y rajoutant plus de neurones ou en choisissant une taille des vecteurs d'entrée/poids plus grande, car ces modifications n'ont pas une grande influence sur la fréquence de fonctionnement maximale.

Hikawa et Maeda ont proposé dans [115] une architecture matérielle massivement parallèle d'une carte auto-organisatrice composée de *256 neurones* avec un vecteur de poids d'une dimension de *3*. Dans cette architecture utilisée pour une application de quantification de couleurs, chaque itération est exécutée en *un seul cycle d'horloge*. La dimension réduite du vecteur du poids et l'utilisation de la distance *Manhattan*, permettent d'optimiser les phases de calcul de la distance ainsi que la mise à jour des poids. D'un autre côté, le *comparateur global* utilisé pour comparer lors d'une itération les distances calculées par tous les neurones pour trouver le neurone gagnant, influence grandement la fréquence de fonctionnement maximale de l'architecture. De plus, elle chute de manière non linéaire avec le nombre de neurones dans la couche compétitive, comme illustré figure 2.8(b). D'autre part, l'augmentation de la taille de la carte de Kohonen engendre l'augmentation de la quantité des ressources matérielles nécessaires comme attendu. Pour une technologie *Field-Programmable Gate Array* (*FPGA*) *Xilinx Virtex 6, XC6VSX315T*, les ressources matérielles exprimées en termes des registres et de *Look-Up Table* (*LUT*), sont présentées figure 2.8(a) en fonction de la taille de la carte *SOM*. La courbe 2.8(b) montre également que la taille de la carte *SOM* exprimée en nombre de neurones dans la couche compétitive permet d'améliorer les performances globales d'apprentissage de la carte auto-organisatrice. À une fréquence de fonctionnement maximale de *33 MHz*, les performances maximales obtenues

sont de *25 344 MCPS/MCUPS* pour les phases d'apprentissage et de rappel. Avec cette architecture, les performances de la phase d'apprentissage et celles de la phase de rappel sont égales, car dans les deux cas, une itération se déroule en un cycle d'horloge.



(b) Ressources matérielles utilisées (b) Performances temporelles de l'impémentation

FIGURE 2.8 – *Résultats d'implémentation d'architecture massivement parallèle*

Nous pouvons constater que les architectures massivement parallèles permettent d'exploiter d'une manière plus optimale le parallélisme de l'algorithme de Kohonen. Différentes architectures matérielles permettent de mettre en avant ce parallélisme inhérent de l'algorithme de Kohonen, au prix d'une latence, une fréquence de fonctionnement maximale et des ressources matérielles nécessaires plus ou moins élevées. Les opérations les plus critiques et les plus influentes sur ces critères sont l'exécution parallèle de l'opération de calcul de la distance L_1 et/ou L_2 et l'opération de recherche du neurone gagnant pour un vecteur d'entrée. Les architectures matérielles présentées jusqu'à présent sont plus adaptées pour des applications à un nombre réduit de neurones avec un vecteur de poids de petites dimensions, car en augmentant le nombre de ces derniers les critères présentés précédemment se dégradent rapidement d'une façon non linéaire. En effet, des cartes de Kohonen de grandes tailles en termes de nombre de neurones sur la couche compétitive ainsi que la dimension des vecteurs d'entrée et de poids, ne permettent pas d'aboutir à des architectures opérant à des fréquences de fonctionnement élevées, engendrant ainsi l'abaissement des performances globales mesurées en *MCPS* et *MCUPS*. Les *architectures matérielles massivement parallèles* à latence plus élevée peuvent relativement stabiliser la fréquence de fonctionnement maximale en fonction du nombre de neurones de la carte de Kohonen, en rallongeant la durée en nombre de cycle d'horloge nécessaire pour effectuer les phases d'apprentissage et de rappel. En effet, le gain obtenu en maintenant une fréquence de fonctionnement maximale la plus stable et plutôt indépendante de la taille de la carte de Kohonen est souvent contrebalancé par les durées nécessaires pour les phases d'apprentissage et de rappel de plus en plus longues. Par conséquent, les performances globales d'une carte de Kohonen souvent n'augmentent pas mais au contraire diminuent pour les tailles de carte de plus en plus importantes. Pour ces raisons, des architectures simplifiées par rapport aux architectures massivement parallèles ont été proposées

dans la littérature, dans le but principal d'implémenter des cartes de Kohonen de grande taille comportant de plus en plus de neurones. Les exemples de ce type d'architecture matérielle sont les architectures sérielle et séquentielle présentées dans les sections suivantes.

2.3.3 Architecture sérielle parallèle de la carte SOM

Pour simplifier l'architecture matérielle massivement parallèle de la carte auto-organisatrice de Kohonen au niveau des ressources matérielles nécessaires ainsi que le chemin critique de l'opération de calcul de la distance, tout en exploitant le parallélisme inhérent de l'algorithme de Kohonen, une architecture matérielle sérielle parallèle a été proposée. Comme dans le cas de l'architecture massivement parallèle, l'architecture sérielle est une architecture parallèle de type *SIMD*. Elle exploite le parallélisme inhérent de l'algorithme de Kohonen en distribuant les neurones de la couche compétitive sur plusieurs modules physiques, de la même manière que dans une architecture massivement parallèle. Dans l'architecture sérielle, chaque neurone est un module élémentaire indépendant qui effectue localement le calcul de la distance, la génération des paramètres d'adaptation et la mise à jour du vecteur des poids. Au niveau global de l'architecture, tous les neurones effectuent leurs opérations de manière indépendante et simultanée. La principale différence d'une architecture sérielle par rapport à une architecture massivement parallèle est au niveau des opérations effectuées au sein d'un neurone. Dans un neurone d'une architecture sérielle, toutes les opérations sont effectuées de manière séquentielle. Les éléments de la couche d'entrée sont envoyés un par un vers les neurones. Au cours d'un cycle d'horloge, le neurone sera réservé pour le traitement des éléments d'indice i (x_i et w_i) des vecteurs d'entrée et de poids X et W. Au prochain cycle, le neurone effectuera le traitement des éléments d'indice *i+1* et ainsi de suite.

Le schéma bloc du circuit de calcul de la distance d'un neurone de l'architecture sérielle est présenté figure 2.9. Ce circuit comporte un certain nombre d'opérateurs arithmétiques placés en série permettant de traiter une simple paire d'éléments (x_i, w_i) à la fois. A la fin de chaque calcul sur une paire d'éléments des vecteurs $\overrightarrow{X}$ et $\overrightarrow{W}$, le résultat est provisoirement stocké dans un registre pour sa réutilisation ultérieure au cours du traitement suivant. Après le traitement des derniers éléments des vecteurs $\overrightarrow{X}$ et $\overrightarrow{W}$, le résultat final est récupéré et sauvegardé pour une comparaison ultérieure. Dans cette architecture, le nombre de cycles d'horloge alloués pour l'exécution d'une opération locale est calculé en fonction du nombre d'éléments sur la couche d'entrée.

La mise en œuvre du circuit de traitement des paires d'éléments de vecteur $\overrightarrow{X}$ et $\overrightarrow{W}$ peut être effectuée d'une façon asynchrone , avec un circuit composé d'une série d'opérateurs arithmétiques et logiques permettant de traiter une paire d'éléments en un seul cycle d'horloge [106, 120] ou synchrone en plaçant une bascule entre les opérateurs arithmétiques et logiques et donc le traitement sera effectué en pipeline sur le circuit [113, 121]. Dans [106], *Lachmair et al.* ont proposé une architecture sérielle parallèle permettant de réaliser une carte auto-organisatrice de grande taille, composée de *6050 neurones* avec des vecteurs d'entrée/poids d'une dimension de *194*. Un

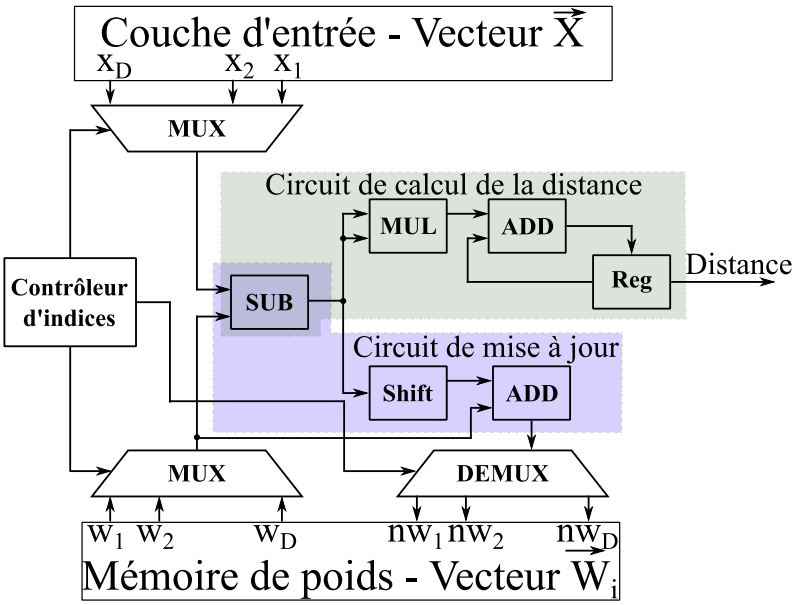


FIGURE *2.9 – Le schéma bloc du circuit de calcul de la distance d'un neurone dans l'architecture sérielle de la carte SOM*

neurone de cette architecture peut être reconfiguré pour calculer soit la distance avec la norme L_1 ou la norme L_2. Les opérateurs du circuit (*soustracteur*, *multiplieur*, *accumulateur*) sont placés d'une façon asynchrone. Le traitement d'une paire d'éléments des vecteurs $\overrightarrow{X}$ et $\overrightarrow{W}$ est effectué en un cycle d'horloge dont la valeur maximale de fréquence est de *99 MHz* sur une technologie *FPGA Xilinx-Virtex-4*. Dans cette architecture, le calcul de la distance L_1 ou L_2, entre les vecteurs de poids et d'entrée composés de *194 éléments*, est effectué en *194 cycles d'horloge*. Par conséquent, les performances d'apprentissage obtenues avec cette architecture s'élèvent à *20 604 MCUPS*.

Afin d'améliorer les performances globales (d'apprentissage et de rappel) du système utilisant une architecture matérielle de type sériel parallèle, *Hendry et al.* ont proposé une architecture sérielle synchrone dans [121]. Dans cette architecture, les opérateurs arithmétiques du circuit de traitement local au sein d'un neurone sont synchronisés par des registres intermédiaires. En utilisant la distance *Manhattan*, le circuit est composé de trois opérateurs arithmétiques : un *soustracteur*, un opérateur de *valeur absolue* et un *accumulateur*. Avec ce traitement pipeline, chaque opérateur traite un élément de vecteur au cours d'un cycle d'horloge. Par conséquent, la distance finale sera calculée après *D+3 cycles* d'horloge dont la fréquence est de *200 MHz* sur une technologie CMOS, avec *D* étant le nombre d'éléments de la couche d'entrée pouvant atteindre *1024*. Pour minimiser la consommation d'énergie au cours de la phase de rappel, un seuil d'accumulation de la distance est également utilisé pour éliminer un certain nombre de neurones de l'élection du neurone gagnant. Ce seuil correspond à la plus grande distance minimale calculée par un neurone. Au cours du calcul de la distance par un neurone, si la valeur dans l'accumulateur dépasse ce seuil fixé à l'avance, le neurone en question ne terminera pas le traitement de tous les éléments (suivants) du vecteur, et il sera ainsi éliminé de la compétition. Par conséquent, à la fin de

l'opération de calcul de la distance, seuls les neurones ayant une distance inférieure au seuil fixé participeront à l'élection du neurone gagnant. Cette opération est effectuée sur un comparateur global similaire au comparateur utilisé dans une architecture massivement parallèle. Quelques travaux dans la littérature ont également proposé des améliorations de ce module au sein d'une architecture sérielle [121, 128].

Les approches basées sur une architecture sérielle trouvées dans la littérature permettent d'optimiser la quantité des ressources matérielles nécessaires pour une implémentation parallèle de la carte auto-organisatrice. Le traitement sériel des éléments des vecteurs $\overrightarrow{X}$ et $\overrightarrow{W}$ permet également d'améliorer la fréquence maximale de fonctionnement même pour des vecteurs d'entrée/-poids de grande dimension. Autrement dit, l'architecture globale est robuste par rapport aux changements de la structure de la couche d'entrée dans une certaine limite définie par les ressources matérielles allouées pour l'architecture. De plus, ces changements de dimension de vecteurs d'entrée au delà des limites allouées initialement ne nécessitent pas de grands efforts de conception pour une adaptation de l'architecture, puisque au niveau du circuit de calcul de la distance aucun changement n'est à faire, uniquement les ressources de stockage des vecteurs de poids et d'entrée doivent être re-dimensionnées.

2.3.4 Architecture séquentielle de la carte SOM

L'architecture séquentielle de la carte auto-organisatrice est une des architectures matérielles les plus simples à mettre en œuvre. Cette architecture, présentée sur la figure 2.10, est composée d'un module de traitement ou de calcul couplé à un comparateur et deux registres : *registre de distance minimale* (R_d) et *registre d'identité du neurone gagnant* (R_{id}). Les vecteurs de poids de tous les neurones de la couche compétitive sont enregistrés dans une mémoire nommée *Mémoire de poids*, ou chaque vecteur est enregistré dans une zone mémoire indexée par l'identité de son neurone associé.

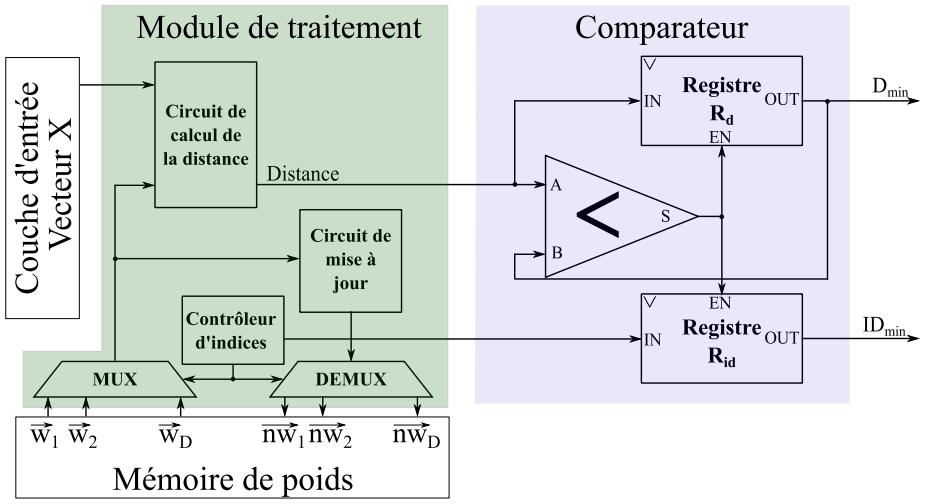


FIGURE *2.10 – Architecture séquentielle de la carte SOM*

Le calcul de la distance L_1 et/ou L_2 de tous les neurones est réalisé par le module de traitement de manière séquentielle. A un instant donné, le module de traitement calcule la distance L_1/L_2 entre le vecteur de poids du neurone d'identité *i* et le vecteur d'entrée. Une fois la distance calculée, elle est envoyée vers le comparateur, tandis que le module de traitement reprend le calcul de la distance L_1/L_2 du neurone d'identité *i+1* et ainsi de suite. Au niveau du comparateur, la distance calculée est comparée avec la distance minimale enregistrée dans le registre R_d. Si la distance dernièrement calculée est inférieure à la distance enregistrée dans le registre R_d, elle la remplacera dans le registre R_d et le registre R_{id} sera mis à jour avec l'identité du neurone correspondant. Donc, dans l'architecture séquentielle le calcul de tous les neurones est distribué temporellement sur un module unique partagé dans le temps, contrôlé par un module de contrôle utilisant les identités de neurones pour sélectionner le calcul associé à un neurone. D'un autre côté, dans une architecture séquentielle, les calculs à réaliser au sein d'un neurone peuvent être effectués soit de manière sérielle, où un élément de vecteur est traité à la fois soit de manière parallèle où tous les éléments de vecteur sont traités en même temps.

Kurdthongmee a proposé dans [129] une architecture matérielle séquentielle où les calculs au sein d'un neurone sont effectués de manière parallèle sur tous les éléments des vecteurs d'entrée et des vecteurs poids. L'architecture présentée a été validée sur des vecteurs d'une dimension de *3*. Les calculs au sein d'un neurone s'effectuent en *un cycle d'horloge* à une fréquence de *50 MHz*. La technologie utilisée est *FPGA Xilinx XC2VP1000*. Cette architecture n'est pas gourmande en ressources matérielles, étant donné l'utilisation d'un seul neurone pour toute la carte *SOM*. De surcroît, les modifications éventuelles du nombre de neurones sur la couche compétitive pour adapter une carte *SOM* à une large gamme d'applications ne nécessitent pas de grands efforts de conception. Il suffit d'élargir la taille des blocs mémoires pour pouvoir rajouter les nouveaux vecteurs de poids et modifier le contrôleur choisissant le neurone à traiter. L'augmentation du nombre de neurones n'influence guère les ressources matérielles utilisées. En effet, les ressources ajoutées consistent aux blocs mémoires ou registres utilisés pour l'enregistrement des vecteurs de poids et au multiplexeur servant pour le contrôle du choix de neurone à traiter.

Une autre architecture séquentielle à base d'un circuit de traitement sériel synchrone a été proposée par *Hikawa et Kaida* dans [116]. Cette architecture traite des vecteurs d'entrée/poids d'une dimension égale à *32*, pour lesquels *34 cycles d'horloge* sont nécessaires, avec une fréquence de *188 MHz* obtenue en utilisant la technologie *FPGA Xilinx XC3S700A*. Comme pour l'architecture présentée [129], le changement de structure de la carte *SOM* basée sur cette architecture, notamment en termes du nombre de neurones ou de dimensions de vecteur d'entrée / poids, est relativement simple où uniquement les supports de stockage doivent être re-dimensionnés. De plus, la particularité de cette architecture est sa capacité de « *culling* » ou d'élagage temporaire de neurones permettant d'améliorer la précision de la carte *SOM* appliquée à la reconnaissance de gestes des mains.

L'architecture matérielle séquentielle est l'architecture la moins couteuse en termes de ressources matérielles utilisées pour sa conception. D'un autre côté, cet avantage en termes de res-

sources nécessaires est au détriment des performances pouvant être obtenues dans les phases d'apprentissage et de rappel. Un autre avantage des architectures matérielles séquentielles de l'algorithme de Kohonen est qu'elles peuvent être adaptées avec peu d'effort à n'importe quelle application. De plus, elles peuvent être conçues de telle sorte pour garantir un certain niveau de flexibilité, une propriété peu commune pour une architecture matérielle lui permettant de s'adapter au cours de son fonctionnement. Comme discuté précédemment, les autres types d'architectures matérielles sont peu voire aucunement flexibles. Ce problème de flexibilité représente la faiblesse principale des architectures matérielles de la carte de Kohonen qui sont principalement caractérisées par de très bonnes performances par rapport à leurs équivalents logiciels ainsi que par leurs coûts faibles en termes de ressources matérielles utilisées.

2.4 Problème de flexibilité des architectures matérielles de la carte de Kohonen

En vue de leur capacité de traitement des données multidimensionnelles les cartes auto-organisatrices sont fréquemment utilisées dans différents domaines tels que traitement d'images, biométrie, reconnaissance, robotique, etc. Une carte de Kohonen est caractérisée par plusieurs paramètres [69, 130] :

Dimension du vecteur de poids : correspond au nombre d'éléments sur la couche d'entrée. Ce paramètre dépend du nombre de valeurs permettant de représenter au mieux un échantillon de l'environnement externe selon caractéristiques associées à une application.

Taille de la couche compétitive : correspond au nombre de neurones sur la carte *SOM*. Ce paramètre dépend de la diversité ainsi que de la densité des données dans l'espace d'entrée. D'autre part, elle dépend également de la précision que l'on souhaite obtenir à base de la carte de Kohonen.

La topologie de voisinage : correspond à la façon d'interconnexion des neurones sur la couche compétitive d'une carte *SOM*. Ce paramètre permet de déterminer le taux de voisinage utilisé au cours de la phase d'apprentissage pour la mise à jour des vecteurs de poids de neurones. Ce paramètre dépend de la densité des données dans l'environnement externe.

Le rayon de voisinage initial : correspond à la zone autour du neurone gagnant dont les neurones seront affectés par l'adaptation et leurs vecteurs de poids seront mis à jour. Ce paramètre dépend du nombre de classes identifiées sur la couche de sorties et est dépendant de la fonction réalisée.

La structure finale d'une carte auto-organisatrice qui sera utilisée au sein d'une application varie en fonction de ces paramètres. Une carte de Kohonen doit respecter les besoins applicatifs ainsi que les caractéristiques des données de l'environnement externe pour pouvoir répondre de la meilleure des manières à la fonction qui lui a été confiée. Une carte *SOM* paramétrée pour une application donnée peut difficilement être utilisée pour une autre. En effet, le changement

d'application ou d'environnement externe imposent le changement de modèle de la carte *SOM* et donc la modification de sa structure et de ses paramètres. Les architectures implantant l'algorithme de Kohonen doivent pouvoir supporter ces changements dynamiques. L'adaptation d'une carte *SOM* avec un jeu de paramètres initial propre à une application à une autre est souvent plus simple à réaliser dans des implémentations logicielles. Dans la littérature, on trouve même des modèles *auto-adaptables* de la carte de Kohonen mis en œuvre à base des implémentations logicielles [94, 131–134].

D'autre part, certains domaines d'application imposent des contraintes temporelles serrées pour assurer un traitement temps réel. Comme discuté précédemment, les performances temporelles obtenues avec ces implémentations logicielles généralement ne permettent pas de répondre à ces exigences. Dans les systèmes embarqués, pour pouvoir augmenter leur autonomie, à ces contraintes de performances s'ajoutent souvent des contraintes de faible consommation d'énergie. Le résultat de toutes ces contraintes sont des architectures matérielles performantes permettant d'exploiter au mieux le parallélisme de l'algorithme de Kohonen et les besoins des applications pour lesquelles elles ont été proposées. D'un autre côté, ces implémentations matérielles de la carte auto-organisatrice dédiées à des applications spécifiques, sont peu voir aucunement adaptables à des changements de paramètres à la volée (*online*) au cours de leur fonctionnement. En cas de changement d'application ou de fonction, afin de répondre à ces nouveaux besoins caractérisés par un nouveau jeu de paramètre, la seule solution est de repasser par toutes les phases de conception pour modifier le modèle de la carte *SOM*. Le manque de flexibilité des implémentations matérielles de la carte *SOM* est une problématique de recherche de longue date. Dans la littérature, on trouve un certain nombre de travaux où les auteurs ont proposé des architectures matérielles reconfigurables permettant d'adapter un certain nombre de paramètres de la carte *SOM* par une simple configuration. Ces architectures matérielles reconfigurables de la carte *SOM* font l'objet de la section suivante.

2.4.1 Architectures matérielles reconfigurables de la carte SOM

Dans [135], *Kolasa et al.* ont proposé une architecture matérielle reconfigurable permettant de configurer la topologie de voisinage utilisée. Dans cette architecture, les connexions interneuronales peuvent être paramétrées pour utiliser une des topologies précédemment présentées : *topologie en losange*, *topologie rectangulaire* ou *topologie hexagonale*. La topologie de voisinage agit sur le nombre de neurones qui participent à la phase d'adaptation. Le choix d'une topologie de voisinage dépend essentiellement de la distribution des données dans l'espace d'entrée et permet de réduire l'erreur de quantification de la carte *SOM* pour un même espace d'entrée. Comme présentée sur la figure 2.11, l'architecture proposée consiste à une grille de neurones connectés par des communications point à point paramétrables entre leurs voisins les plus proches physiquement. Le maillage physique initial de neurones utilisé dans cette architecture est la topologie *rectangulaire* comme illustré la figure 2.11. Pour changer de topologie, un simple paramétrage de système active le circuit de contrôle des communications au niveau de chaque neurone qui se

charge de la désactivation des communications inutiles pour la nouvelle topologie. Trois exemples de configuration différente sont présentés figure 2.11(a), (b) et (c).

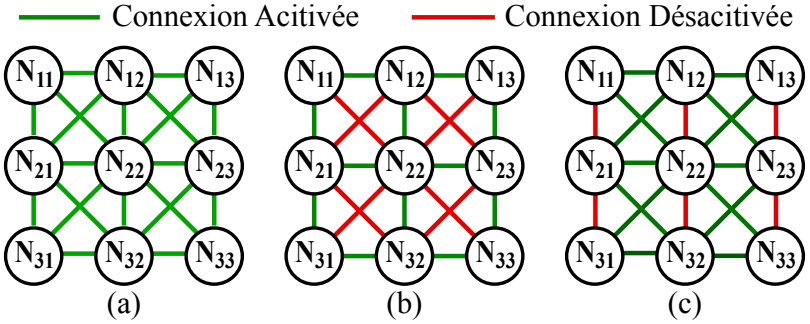


FIGURE *2.11 – Architecture matérielle reconfigurable de la carte SOM en topologie de voisinage : (a) configuration rectangulaire (b) configuration en losange et (c) configuration hexagonale*

Tamukoh et al. ont proposé dans l'article [119], une architecture matérielle reconfigurable en termes de la distance L_1 ou L_2 et des fonctions de voisinage utilisées. Cette architecture peut être configurée pour calculer la distance selon la norme L_1 ou la norme L_2. D'autre part, la fonction de voisinage peut être choisie entre la fonction de voisinage classique ou la fonction *RWTA* détaillée précédemment [118], via une simple reconfiguration. Par conséquent, en combinant les deux reconfigurations, on peut choisir entre 4 modèles de la carte auto-organisatrice. La reconfiguration de cette architecture matérielle est assurée par la *reconfiguration dynamique* des circuits programmables *FPGA*.

Dans les travaux de *Fengwei et al.* présentés dans [113], une architecture matérielle dynamiquement reconfigurable pour l'implémentation d'un réseau de Kohonen dans sa version supervisée *LVQ* a été proposée. Les auteurs ont exploité la flexibilité des architectures séquentielles à base de neurone effectuant les calculs de manière sérielle pour proposer une architecture flexible en termes de nombre de neurones ainsi que en termes de la dimension de vecteurs d'entrée. Pour améliorer les performances de ce type d'implémentation matérielle, les neurones de la couche compétitive ont été distribués sur plusieurs circuits. Chaque circuit est doté d'une *mémoire RAM* dans laquelle les vecteurs de poids de neurones associés sont enregistrés. Un élément du vecteur de poids appartenant à un neurone est identifié par sa propre adresse dans cette mémoire. Ces adresses sont générées par un bloc de contrôle qui est paramétrable et permet la modification de la dimension des vecteurs d'entrée/de poids. Les distances seront calculées sur chaque circuit en parallèle et envoyées vers un comparateur global qui procédera à une comparaison séquentielle distance par distance au niveau de toute la carte *LVQ*. La comparaison est également contrôlée par un bloc de contrôle paramétrable permettant de modifier le nombre total de neurones de la carte *LVQ* et ainsi la taille de la couche compétitive. Enfin, l'identité du neurone gagnant sera connue après le balayage et la comparaison de toutes les distances envoyées par tous les neurones de la carte *LVQ*. La flexibilité proposée par cette architecture matérielle est au détriment des performances obtenues. En effet, les performances d'apprentissage et de rappel sont respectivement de *592 MCUPS* et *596 MCPS* obtenues sur une carte *LVQ* de taille de *512 neurones* adaptée pour

des vecteurs d'entrée/de poids d'une dimension de *4096*. Ces résultats sont de l'ordre des performances que l'on obtient avec des architectures logicielles et sont plutôt insatisfaisants pour une implémentation matérielle dédiée à un système temps réel à des contraintes de temps fortes. D'autre part, la taille maximale de la couche compétitive avec cette architecture est fixée au préalable pendant la conception et ne peut aucunement être modifiée au cours du fonctionnement. Autrement dit, l'extensibilité à la volée de cette architecture est impossible. En effet, si une modification architecturale est souhaitée pour une application donnée, par exemple un nombre de neurones plus grand ou une dimension de vecteurs de poids plus élevée, le seul moyen d'y parvenir est de passer à nouveau par les phases de conception pour apporter ces modifications sur l'architecture initiale.

2.4.2 Problèmes d'extensibilité des architectures matérielles de la carte SOM

Le problème d'extensibilité évoqué dans la section précédente est un problème existant dans toutes les implémentations matérielles reportées dans la littérature [136, 137]. En effet, l'extensibilité d'une implémentation matérielle de la carte auto-organisatrice consiste à rajouter à la volée (*online*) un ensemble de neurones sur la couche compétitive initiale sans changements architecturaux majeurs. Les neurones nouvellement rajoutés à une carte *SOM* initiale doivent pouvoir être combinés avec les neurones initiaux, pour constituer au final une nouvelle structure de la carte *SOM* dont le nombre de neurones total sera égal à la somme des neurones initiaux et rajoutés. La figure 2.12(a) présente le schéma synoptique d'une implémentation matérielle massivement parallèle de la carte auto-organisatrice dite l'architecture « classique ». Dans cette implémentation, la carte est composée d'un ensemble de neurones distribués sur une grille composée de *L* lignes et de *K* colonnes. Tous les neurones sont alimentés par les éléments de la couche d'entrée par des liens de communication point à point. Autrement dit, la couche d'entrée comportant le vecteur d'entrée est connectée à tous les neurones de la carte *SOM*. Dans cette architecture, chaque neurone envoie la distance calculée entre son vecteur de poids et le vecteur d'entrée également via des liens de communications point à point, vers un module de recherche du neurone gagnant. Ce module est équipé de $L \times K$ entrée pour récupérer les distances calculées par tous les neurones et leurs identités. La distance minimale et l'identité correspondante, seront dirigées vers les $L \times K$ sorties du module de comparaison pour informer, également via des liens de communication point à point, tous les neurones de la carte de l'identité du neurone gagnant.

Dans l'objectif d'avoir des architectures matérielles de la carte *SOM* configurables et adaptables à toute sorte d'application, la question que l'on pourra se poser l'architecture matérielle classique de la figure 2.12(a) comportant $\mathrm{L}\times\mathrm{K}$ neurones pourra-t-elle servir de base pour construire une autre architecture comportant plus de neurones, $\mathrm{M} \times \mathrm{N}$ comme illustré figure 2.12(b) ? La réponse à cette question est malheureusement négative. Plusieurs problèmes sont à l'origine de cette situation : le problème de comparaison de distances calculées par tous les neurones, le problème d'identification unique d'un neurone au sein de la nouvelle carte *SOM* et le problème de distribution des vecteurs d'entrée vers tous les neurones. Le problème principal réside dans le module de

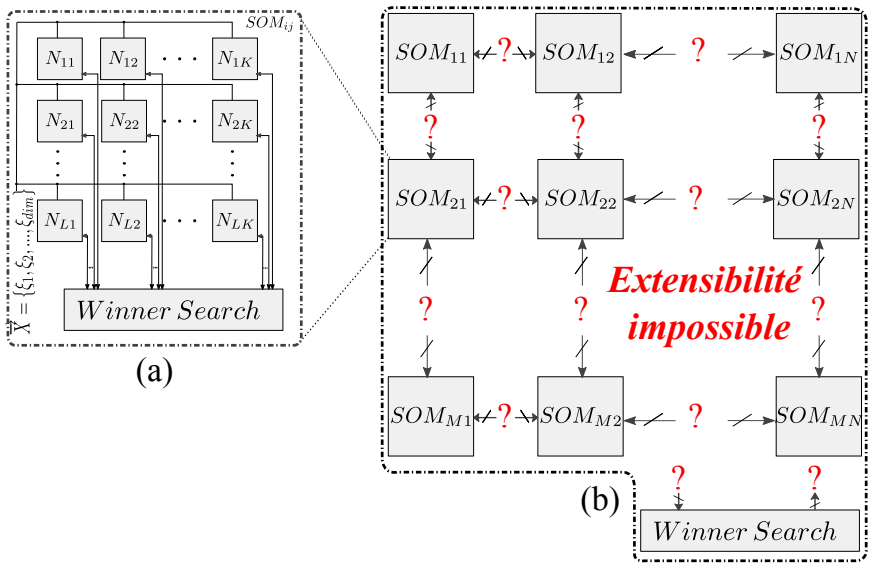


FIGURE 2.12 – *Problème d'extensibilité des architectures matérielles dites classiques de la carte auto-organisatrice : (a) Architecture d'une carte SOM classique de taille* $\mathrm{L} \times \mathrm{K}$. *(b) Extensibilité de la carte SOM : essai de combinaison de* $\mathrm{M} \times \mathrm{N}$ *cartes SOM de taille* $\mathrm{L} \times \mathrm{K}$.

comparaison dont le nombre d'entrées et de sortie ne permet pas d'ajouter de nouveaux neurones à la compétition, puisque ce nombre est figé dans la phase de conception. De plus, ce module de comparaison ne permet pas, sans modifications majeures dans la phase de conception, d'être combiné de manière directe avec d'autres modules de comparaison appartenant à d'autres architectures indépendantes de la carte *SOM*. Un autre problème dans cet exercice d'extensibilité est le problème d'identification des neurones dans la nouvelle carte *SOM* formée. Sans adaptation d'identifiants de neurones appartenant à des plus petites cartes *SOM*, il y aura des neurones de la nouvelle grande carte *SOM* dont les identités ne seront pas uniques et ceci pourra compromettre le bon déroulement de l'élection du neurone gagnant et ainsi la bonne implantation de l'algorithme de Kohonen.

Dans la littérature, nous trouvons également des architectures matérielles de la carte *SOM* utilisant une approche de comparaison globale différente de celle évoquée précédemment notamment la comparaison sérielle et distribuée [114, 118, 138]. L'approche de comparaison sérielle, généralement utilisée dans les architectures matérielles séquentielles, permet d'identifier le neurone gagnant en prenant les distances calculées par les neurones de la carte *SOM* une par une. Cette approche permet de garantir un certain niveau de flexibilité [116, 129], mais au détriment des performances d'apprentissage et de rappel. D'un autre côté, l'exploitation de cette approche de comparaison ne permet pas non plus de garantir un degré d'extensibilité souhaité dans les architectures matérielles de la carte *SOM* [113]. La solution des comparateurs locaux distribués est une autre approche de recherche du neurone gagnant au sein d'une carte *SOM*. Ces comparateurs distribués effectuent l'opération de recherche du neurone gagnant par propagation *systolique* des distances minimales locales sur des étages [55, 139]. De cette façon, les comparaisons sont effectuées séquentiellement étage par étage avec une possibilité d'effectuer des comparaisons locales

en parallèle sur des comparateurs situés sur le même étage systolique. L'architecture d'une carte auto-organisatrice utilisant ce mécanisme de *propagation systolique* de distance minimale est présentée sur la figure 2.13. Chaque comparateur associé à un neurone assure une comparaison entre sa distance calculée et les distances minimales locales envoyées par ses voisins adjacents situés dans l'étage précédent, en amont de la propagation systolique. Dans cette approche, l'identité du neurone gagnant sera connue dans le neurone placé à l'intersection de la ligne *L* avec la colonne *K* (l'étage T_{K+L-1} de l'architecture présentée figure 2.13).

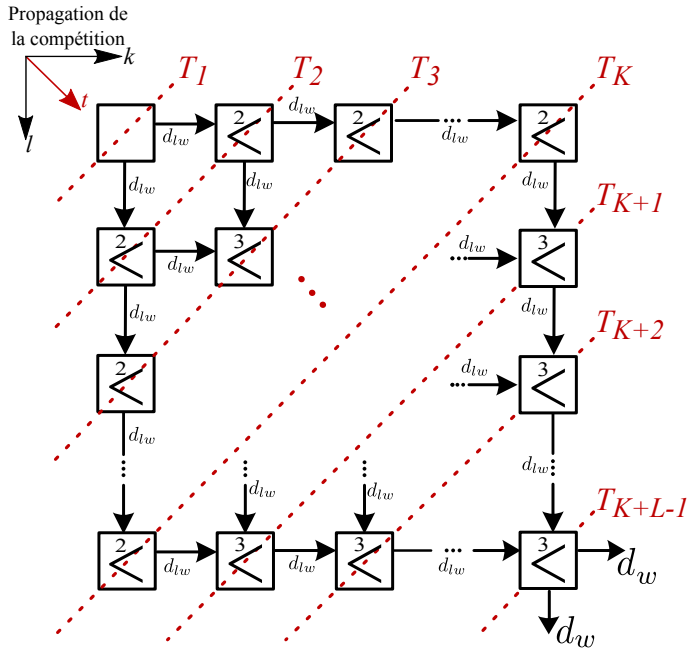


FIGURE *2.13 – Comparateurs distribués avec propagation systolique pour la recherche du neurone gagnant au sein d'une carte SOM*

Cette approche de recherche du neurone gagnant au sein d'une carte *SOM*, permet d'offrir un certain niveau de flexibilité et pourrait même offrir un certain niveau d'extensibilité aux implémentations matérielles avec des modifications mineures. En effet, un simple paramétrage des connexions d'un neurone pourrait offrir la possibilité de poursuivre cette propagation systolique même au sein des cartes *SOM* d'une taille plus grande. D'un autre côté, le principal frein à cette extensibilité sont les liens de communication point à point qui sont figés et doivent être prévus à la phase de conception pour pouvoir interconnecter plusieurs cartes *SOM*.

Les communications de type point à point ne garantissent pas la flexibilité d'échange de données en cas de modification de la structure de la carte auto-organisatrice. En fait, les communication point à point sont utilisées pour interconnecter deux unités de calcul *A* et *B* directement via un lien physique. Cette communication figée est reliée d'un côté à l'entrée de l'unité *A* et à la sortie de l'unité *B* de l'autre côté. Le comportement de cette communication dépend de traitement des unités. Par conséquent, les communications utilisées pour l'implémentation matérielle d'une carte auto-organisatrice dépendent de son fonctionnement et donc de sa structure. Dans ce cas, le changement de l'application qui exige le changement de la structure de la carte, demande une nouvelle

conception pour adapter les communications à la nouvelle structure. Une solution a été proposée pour faciliter la conception de la carte auto-organisatrice. Cette solution consiste à proposer des architectures génériques [140, 141]. En cas de changement d'application, un simple paramétrage permettra de changer la structure de la carte. Cette reconfiguration permettra de fixer la structure adaptable pour la nouvelle architecture. Donc, les communications point à point seront fixées à base du comportement souhaité de la carte selon la nouvelle structure.

TABLEAU *2.1 – Comparaison des approches d'implémentation de la carte auto-organisatrice de Kohonen*

Approche	Flexibilité	Extensibilité	Ressources matérielles	Performance temporelle
Logicielle [93,94]	++	++	-	–
Logicielle parallèle [95,97,105]	+	+	–	-
Matérielle Massivement parallèle [111,115,125]	–	–	-	++
Matérielle séquentielle parallèle [110]	-	+	+	-
Matérielle séquentielle sérielle [116]	+	+	++	–
Matérielle sérielle parallèle [106,119]	-	–	+	+
Matérielle systolique [139]	-	++	+	+
Matérielle reconfigurable [113,119,135]	+	-	+	+
Matérielle générique [140]	+	+	-	++

Après l'analyse des différentes approches d'implémentation de la carte auto-organisatrice de Kohonen proposée dans la littérature, la synthèse de cette étude bibliographique est présentée dans le tableau 2.1.

2.5 Conclusion

Dans ce chapitre, nous avons passé en revue toutes les architectures permettant de mettre en œuvre les cartes de Kohonen sur des supports dits logiciels ou matériels. Les architectures logicielles de la carte de Kohonen sont caractérisées par une flexibilité accrue permettant de les adapter plus facilement à une large gamme d'application des domaines différents. D'un autre côté, cette flexibilité est au détriment des performances en phases d'apprentissage et de rappel, pouvant limiter l'utilisation des architectures logicielles de la carte *SOM* dans les applications temps réel à fortes contraintes de temps. D'autre part, les performances des architectures matérielles de la carte *SOM* sont souvent au rendez-vous pour les applications temps réel, mais cette fois au détriment de la flexibilité étant faible presque inexistante au sein de ces architectures. Dans cette section, nous nous sommes intéressés en particulier aux architectures matérielles de la carte *SOM* et avons identifié les mécanismes principaux étant à l'origine de leur faible flexibilité : la recherche du neurone gagnant étant souvent centralisée, l'impossibilité d'interconnecter de cartes *SOM* existantes pour en former des cartes *SOM* de plus grande taille - le manque d'extensibilité, et les connexions entre neurones et blocs principaux d'une carte *SOM* étant de manière générale réalisées par des liens physiques de type point à point permettant de connecter directement

tous les modules concernés. Dans la section suivante, nous nous intéressons en particulier aux moyens différents de communications utilisés dans les architectures *SOM* et dans les architectures matérielles de manière générale, et allons proposer une nouvelle approche matérielle de la carte *SOM* offrant par rapport aux architectures matérielles existantes la propriété d'extensibilité tout en gardant les performances caractéristiques des approches matérielles de la carte *SOM*.

Chapitre 3

Architecture matérielle Adaptable de la carte auto-organisatrice

Sommaire

3.1 Introduction

Comme nous l'avons vu dans les chapitres précédents, la *carte auto-organisatrice de Kohonen* se présente comme une des approches les plus utilisées dans la famille des réseaux de neurones artificiels à apprentissage non-supervisé. Plusieurs implémentations logicielles et matérielles de la carte auto-organisatrice ont été proposées dans la littérature. Les implémentations logicielles étant relativement simples à mettre en œuvre offrent un très bon niveau de flexibilité permettant d'adapter dynamiquement la structure et le comportement de la carte auto-organisatrice aux besoins applicatifs. D'autre part, les implémentations matérielles permettent d'exploiter le parallélisme inhérent de *l'algorithme* SOM *de Kohonen* en offrant des meilleurs performances par rapport aux implémentations logicielles et en réduisant la consommation d'énergie utilisée. De plus, les implémentations matérielles sont plus adaptées pour des applications ayant des contraintes temps réel fortes ne pouvant pas être satisfaites avec des solutions logicielles.

Les implémentations matérielles de la carte auto-organisatrice proposées dans l'état de l'art, sont souvent dédiées à des applications spécifiques. Les principaux paramètres d'une carte auto-organisatrice sont définis au cours de la phase de conception du système, selon les besoins applicatifs. Donc, l'architecture conçue conviendra parfaitement aux besoins de l'application spécifique. Par contre, si l'environnement dans lequel l'application évolue ainsi que la fonction réalisée changent, l'adaptation des paramètres de la carte ne pourra pas se faire dynamiquement, et une nouvelle architecture prenant en compte les nouveaux besoins de l'application devra être conçue avant de reprendre l'exécution.

L'inconvénient majeur de ces implémentations matérielles réside dans leur inflexibilité ne leur permettant pas de répondre aux nouvelles exigences imposées soit par l'environnement externe de l'architecture dans lequel elle évolue soit par les nouveaux besoins applicatifs apparaissant au cours de l'exécution à un moment donné. Autrement dit, une architecture matérielle de la carte *SOM* est taillée pour une application spécifique et ne tolère aucune modification au cours de son exécution lui permettant de s'adapter dynamiquement en ligne (online) aux changements survenus. La raison principale de cet inconvénient est l'utilisation des approches de conception classiques principalement basées sur la technique de communication *point à point* ou «*Point-to-Point* (*P2P*)» et résultant en architectures statiques. Bien que cette technique de communication offre des performances temporelles optimales en assurant souvent une communication entre deux modules en un seul cycle d'horloge, elle devient de plus en plus complexe et représente le vé-

ritable goulot d'étranglement avec un nombre croissant de neurones ainsi que la diversité des types de messages échangés. De plus, cette approche est le facteur de limitation principale pour la conception des cartes *SOM* de très grandes tailles.

Pour résoudre le problème de flexibilité des implémentations matérielles des cartes auto-organisatrices et permettre l'adaptation dynamique à la volée (online) de leurs principaux paramètres, des nouveaux paradigmes de conception d'architecture doivent voir le jour. Le facteur majeur limitant la flexibilité identifié dans la discussion précédente est la technique de communication *P2P*. Donc, la première étape consiste à trouver des solutions permettant de remplacer cette technique de communication par une nouvelle approche de communication plus flexible et plus adaptée à des éventuelles évolutions au cours du fonctionnement.

Dans ce chapitre, nous nous intéressons aux différents moyens de communication existants pour la conception des architectures matérielles de manière générale, et allons choisir le moyen de communication le plus flexible nous permettant de proposer une nouvelle approche de conception d'une architecture matérielle de la carte *SOM*. Cette architecture matérielle de la carte *SOM* qui sera proposée dans ce chapitre, offre par rapport aux architectures matérielles existantes la propriété d'extensibilité et de flexibilité de changement de ses principaux paramètres tout en gardant les performances caractéristiques des approches matérielles existantes.

3.2 Les types de communications intra-puce

Dans un système sur puce, les interconnexions entre les modules fonctionnels et les ressources partagées peuvent être réalisées à base de différentes techniques [12]. On distingue trois types de communication :

— Communication point à point.
— Communication par bus partagé.
— Communication à base des réseaux sur puce.

Le choix d'utilisation d'une technique dépend des besoins du système en termes de communication. Chaque technique à ses caractéristiques recommandant son utilisation. Ci-dessous nous présentons brièvement les principales caractéristiques de chaque technique de communication.

3.2.1 Communication point à point

Une communication entre deux modules peut être effectuée tout simplement via des fils ou connexions directs. C'est le principe d'une communication *point à point* (*Point to Point P2P* en anglais) [142]. A base de ces connexions, les modules peuvent échanger des données de manière directe : le sens de propagation du flux de données est unique sur une connexion [143]. Cette technique est simple à réaliser et les protocoles d'échange de données adoptés par cette technique sont d'une faible complexité.

Cette technique de communication est principalement utilisée dans des systèmes nécessitant un débit de transfert de données très élevé. Les systèmes utilisant cette technique de communi-

cation sont relativement simples comprenant un nombre limité de modules ou un nombre réduit de connexions entre modules. L'augmentation du nombre de connexions entre modules d'un système utilisant cette technique de communication engendrerait non seulement une complexité plus élevée au niveau système mais également au niveau des modules eux-mêmes [144].

En général, les propriétés d'une connexion point à point sont spécifiées par le concepteur en fonction des caractéristiques des données à échanger entre les modules du système. Par conséquent, les changements de structure d'un tel système pouvant survenir par exemple en ajoutant un ou des modules au processus d'échange de données, ou même en modifiant les connexions entre les modules source et destination d'un message, ne peuvent pas être effectués d'une façon dynamique, à la volée au cours du fonctionnement du système.

En conclusion, la technique de communication point à point est une technique simple à mettre en œuvre permettant d'obtenir des débits de transfert de données très élevés. Les systèmes l'utilisant sont souvent des systèmes dont le nombre de modules fonctionnels est relativement faible car ce type d'échange de données limite non seulement leur flexibilité, qui est souvent inexistante au sein de tels systèmes, mais également leurs performances globales pour un nombre de modules important.

3.2.2 Communication par bus partagé

La technique de communication par bus partagé ou bus hiérarchique est le type de communication majoritairement utilisé dans les systèmes sur puce [144, 145]. Selon les caractéristiques des modules inter-connectés via le bus partagé, il peut être classé en trois catégories [12] :

- **Le bus backplane** - utilisé pour interconnecter des modules fonctionnels, des mémoires et des blocs de gestion des entrées/sorties.
- **Le bus Processeur-Mémoire** - caractérisé par une large bande passante garantissant un bon débit d'échange de données entre des modules fonctionnels et des mémoires [146].
- **Le bus Entrée-Sortie** - utilisé pour la communication d'un module avec les entrées/sorties. Il est caractérisé par une bande passante réduite par rapport aux autres types de bus. Par conséquent, il peut être connecté au bus « *backplane* » sous forme de bloc de gestion des entrées/sorties pour former un bus hiérarchique. Ce type de bus est généralement utilisé dans des dispositifs industriels, où différents types d'interfaces d'entrée/sortie sont utilisés pour échanger avec le système principal.

Le principe de cette technique de communication consiste à partager une seule connexion par tous les modules appartenant à la même hiérarchie de communication du système [144]. Un bus partagé ou hiérarchique tel qu'il est présenté sur la figure 3.1, est composé d'un ensemble de lignes de connexions permettant la circulation bidirectionnelle des flux de données. Ces connexions permettent l'échange des données entre les différents modules suivant un protocole de communication. La gestion de la circulation des données sur un bus est assurée par un module appelé arbitre de bus. Ce module est connecté à tous les modules du système par des

connexions de contrôle permettant de veiller au respect du protocole utilisé [147]. Dans ce type d'architecture, le module d'arbitrage alloue à la demande le bus à une communication entre un module source (désignée comme un *maître*) et un module destination (désignée comme un *esclave*), les deux connectés au bus en question. Une seule communication peut être effectuée à un moment donné. Une fois la communication est établie entre les deux modules, le module maître initie les échanges avec le module esclave en respectant le protocole de communication utilisé sur le bus en question.

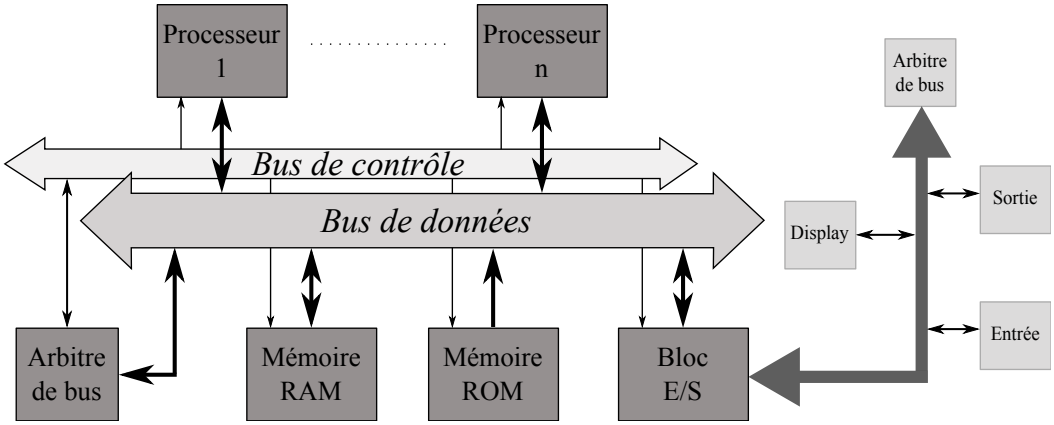


FIGURE *3.1 – Exemple d'architecture de système utilisant comme technique de communication un bus partagé ou hiérarchique*

Un système sur puce utilisant comme technique de communication un bus partagé est caractérisé par un faible coût de réalisation et faibles performances étant donné la distribution temporelle des opérations de communication sur le bus en question. Ces dégradations de performances empirent avec l'augmentation du nombre de modules partageant le bus. En effet, les effets parasites tels que les effets *capacitifs* et *résistifs*, le couplage entre lignes - *crosstalk* ainsi que la consommation d'énergie sont mis en exergue par une longueur de plus en plus élevée des lignes partagées et deviennent significatifs. De plus, cela engendre des retards de propagation sur le bus qui agissent négativement sur la fréquence de fonctionnement du système, pouvant ainsi conduire à son dysfonctionnement.

D'autre part, la complexité de l'arbitrage sur un bus partagé augmente également avec le nombre d'interconnexion et de modules connectés, rendant ainsi son utilisation plus compliquée dans de tels systèmes. Même si ce type de communication est beaucoup plus flexible et par ailleurs plus extensible que son homologue *P2P*, il n'est pas adapté pour des systèmes ayant des grands besoins en communication et comportant un nombre élevé de modules communicants. De surcroit, des changements dynamiques des schémas de communication au cours du fonctionnement d'un système à bus partagé sont pratiquement impossibles, le rendant ainsi inadapté pour des systèmes dynamiques dont la structure évolue au cours du fonctionnement.

TABLEAU *3.1 – Comparaison des techniques de communication*

Propriété \ Technique	P2P	Bus partagé	NoC
Flexibilité	Très mauvais	Bon	Très bon
Extensibilité	Très mauvais	Bon	Très bon
Performance	Très bon	Mauvais	Très bon
Consommation	Très bon	Mauvais	Bon

3.2.3 Communication à base des réseaux sur puce

La technique de communication à base des *réseaux sur puce*, en anglais *Network-on-Chip* (*NoC*), représente une technique de communication alternative permettant de résoudre le problème d'un grand nombre de modules communicants et leurs problèmes d'interconnexions. L'architecture d'un réseau sur puce est une architecture distribuée composée de modules de communication appelés routeurs. Dans cette architecture, un module de calcul tel qu'un processeur, une IP spécifique ou autre d'un système sur puce est associé de manière générale à un routeur. Pour former le système de communication, tous les routeurs sont inter-connectés par des lignes de connexion assurant le transfert des messages sur le réseau suivant un protocole de routage pouvant être spécifié au niveau des routeurs ou au niveau du réseau global.

Cette technique de communication propose une solution d'échange de données permettant de dépasser les problèmes de flexibilité et d'extensibilité des techniques citées précédemment, des bus partagés et des connexions point à point [148–150]. C'est pour ces raisons que les réseaux sur puce représentent la seule solution de communication viable pour les systèmes sur puce à Multiprocesseurs « *Multi-Processor System-on-Chip* (*MP-SoC*) » : une large bande passante, un degré de modularité important, une architecture distribuée, une flexibilité et extensibilité élevées sont parmi les capacités offertes aux systèmes adoptant les réseaux sur puce [151, 152].

Le choix de la technique de communication à utiliser doit être effectué selon les besoins du système sur puce à concevoir. Chaque technique de communication présente un certain nombre d'avantages et d'inconvénients. Une récapitulation comparative, présentée dans le tableau 3.1, résume les propriétés des trois techniques de communication [12].

Aujourd'hui, la tendance dans la conception des systèmes sur puce est vers des architectures distribuées, généralement composées de plusieurs modules fonctionnels opérant en parallèle et permettant ainsi d'effectuer n'importe quel traitement ou calcul complexe en le découpant et distribuant sur ces unités fonctionnelles de calcul fortement connectées. Dans ce type d'architecture, la technique de communication qui s'impose inévitablement par les besoins forts en communication sont les réseaux sur puce.

3.2.4 Les réseaux sur puce

Les *NoC* représentent une projection du concept des réseaux de communication informatiques (*large-scale networking*) dans le domaine des systèmes embarqués intégrés sur une puce (*System-on-Chip (SoC) et MP-SoC*). La figure 3.2 présente l'architecture d'un réseau sur puce. Ce dernier est composé de plusieurs modules de communication distribués appelés « *Routeurs* ». Tous les routeurs communiquent via des « *canaux d'interconnexion* » permettant la circulation bidirectionnelle des données entre deux routeurs voisins. Chaque routeur est associé à un élément de calcul ou *Processing Element* (*PE*). Un *PE* peut être soit un processeur de traitement, une fonction spécifique de calcul (une *Intellectual Property* (*IP*)), soit une mémoire ou même un ensemble de modules de calcul interconnectés entre eux dans une hiérarchie locale. Le dialogue entre un routeur et son *PE* associé est assuré par une interface réseau ou *Network Interface* (*NI*). Le rôle de cette interface est de décoder les informations en entrée du *PE* associé en provenance des autres *PE* du réseau, et de coder celles générées par le *PE* local et destinées à d'autres *PE*.

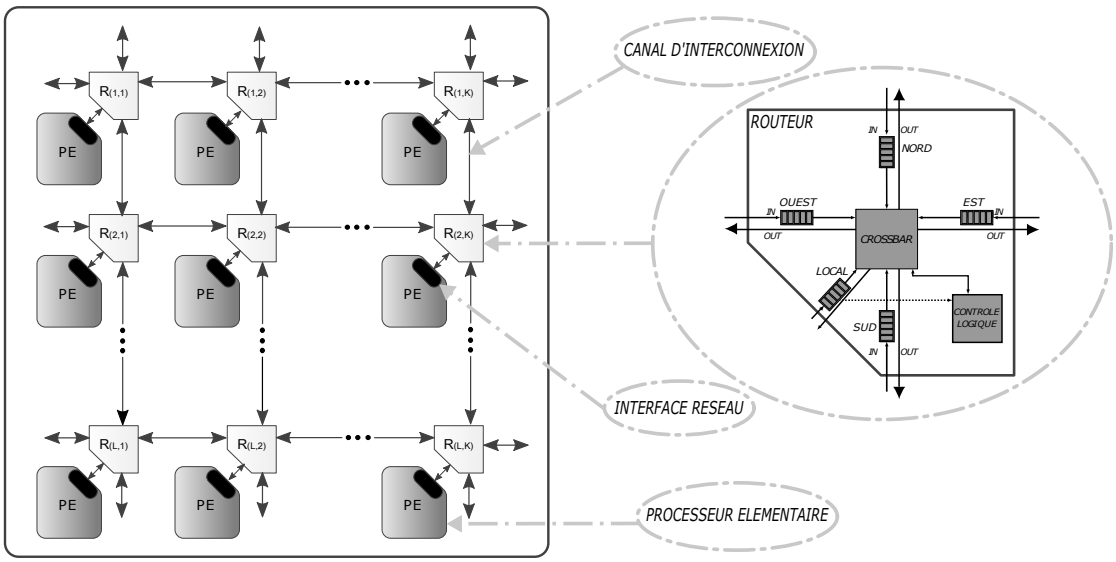


FIGURE *3.2 – Architecture d'un réseau sur puce (NoC) de type 2D maillé*

Le routeur est l'unité principale dans un *NoC*. Il assure la réservation dynamique des circuits de communication inter modules. En effet, un routeur par exemple d'un *NoC 2D maillé*, peut recevoir des données provenant de ces voisins adjacents de quatre côtés : *Nord, Est, Sud* et *Ouest* ainsi que de son *PE* associé. Ces données sont présentées sur une entrée comportant un « *Buffer* » dont le rôle est de stocker provisoirement les données avant leur acheminement final. Ensuite, selon la destination d'un message, les données lui appartenant seront redirigées vers une sortie via une entité « *CROSSBAR* » et contrôlée par un « *contrôleur logique* ». Cette entité intègre un *module d'arbitrage* permettant d'éviter les collisions des données attribuées à la même sortie du routeur à un instant donné.

D'autre part, l'interface *NI* joue un rôle très important dans la procédure de communication des routeurs d'un *NoC*. Ses opérations de codage et décodage permettent la traduction des mes-

sages circulant entre les routeurs sous forme d'information compréhensible par les *PE*. En effet, les informations circulant dans un *NoC* sont présentées sous forme de messages identifiés par l'adresse de leurs destinations. La figure 3.3 présente la structure et la décomposition d'un message. Ce dernier est composé de plusieurs paquets. Pour cela les opérations de codage et décodage sont souvent appelées *paquetage et dépaquetage*. Chaque paquet est divisé en unités de contrôle appelées *flow control units (flits)*. Ensuite, un *flit* est représenté sur le lien physique de communication par plusieurs paquets élémentaires appelés *phits* (*physical units*). En général, le canal d'interconnexion et le *phit* ont la même taille physique en termes de nombre de bits. D'où, un *phit* peut être transféré sur un canal de communication en *un seul cycle d'horloge*. La façon de présenter les informations sous forme de plusieurs de ces unités de données varie en fonction de l'architecture choisie du *NoC* ainsi que de l'algorithme de routage adopté. En fait, le choix de la taille des *paquets*, *flits* et *phits* influe sur le coût, la consommation ainsi que les performances du *SoC*.

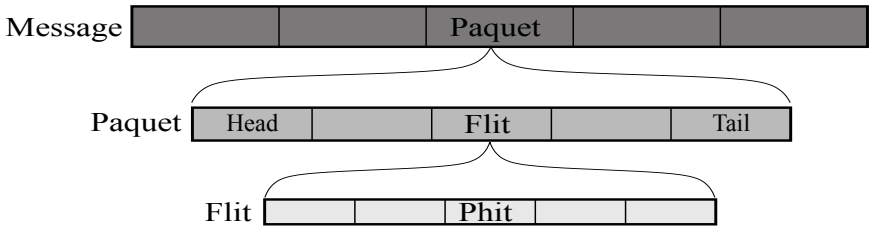


FIGURE *3.3 – Structure d'un message circulant au sein d'un réseau sur puce (NoC)*

i) Réseaux sur puce selon le modèle de référence OSI

Les réseaux sur puce sont conçus sur le modèle universel *Open Systems Interconnection* (*OSI*). La référence universelle de l'organisation des réseaux locaux décompose la procédure de transfert de données en sept couches superposées réalisant une interface entre l'application et le système de transmission de données. La projection du modèle *OSI* sur les réseaux sur puce comporte trois grandes couches du modèle de référence :

- **La couche application :** Cette couche est composée des couches application, présentation et session.
- **La couche architecture :** Cette couche regroupe les couches transport, réseau et liaison des données.
- **La couche circuit :** Cette couche représente la couche physique.

La figure 3.4 présente la projection du modèle *OSI* sur l'architecture des réseaux sur puce. Les couches basses (circuit et architecture) sont responsables de la circulation de l'information sur le réseau. Par contre, la couche haute (application) définit les opérations de communication entre les modules du système intégré, en fonction des besoins de l'application.

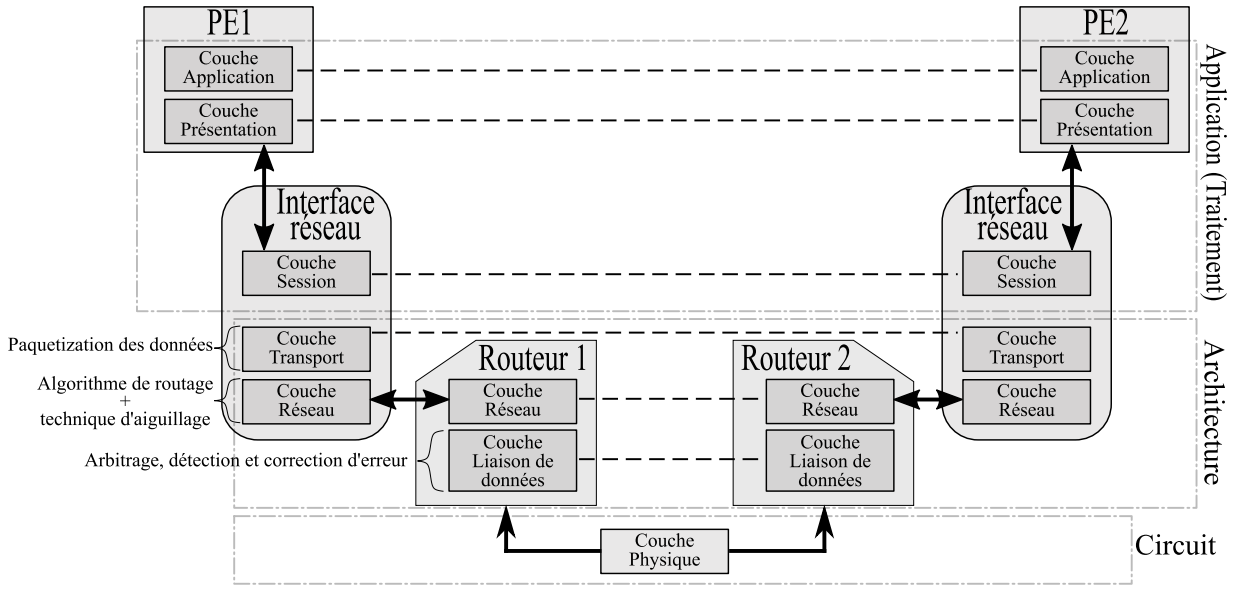


FIGURE *3.4 – Projection du modèle OSI sur les réseaux sur puce*

D'autre part, la distribution des couches sur les différentes unités composant un *NoC* à deux nœuds est ainsi détaillée en figure 3.4. Les couches intégrées dans les entités « *PE* » et « *NI* » (*application, présentation, session, transport* et *réseau*) participent dans une communication entre deux nœuds (la source et la destination), même si ces derniers ne sont pas voisins. Par contre, les couches intégrées dans le routeur ainsi que la couche physique présentée par le canal d'interconnexion, interviennent seulement dans la communication entre deux routeurs adjacents [153].

La couche la plus importante, dans la conception d'un réseau sur puce, est la couche architecture. Cette couche est composée de quatre sous-couches distribuées sur le *NI* et le routeur. La couche *liaison de données*, présente sur le routeur, assure la transmission des paquets d'un routeur à un autre et la gestion des acquittements envoyés par la cible. Cette couche doit être capable de détecter les problèmes de communication sur le réseau et de les corriger. D'autre part, elle assure l'arbitrage de transmission des paquets pour éviter les collisions. Le fonctionnement de cette couche est assurée par l'entité « *Contrôle logique* » (voir figure 3.2). La couche réseau est présente sous forme de deux sous-couches. La première sous-couche au niveau de routeur assure la redirection des paquets suivant la *technique d'aiguillage*. Cette fonction est assurée par l'entité « *CROSSBAR* » (voir figure 3.2). A ce niveau, les données traitées sont sous forme de *phits*. La deuxième sous-couche au niveau de l'interface réseau permet d'ajouter les informations de contrôle au paquet pour assurer le bon fonctionnement de l'algorithme de routage adopté. Les données traitées par cette sous-couche sont sous forme de *flits*. Ainsi, pour chaque *flit* des informations de contrôle doivent être attribuées (l'adresse de destination, le type de *flit*, etc). La dernière sous-couche dans le groupe architecture est la couche transport. Cette couche assure le découpage des données d'un paquet afin de le présenter sous forme de plusieurs *flits*.

ii) Techniques d'aiguillage - *switching techniques*

Les *techniques d'aiguillage* (*Switching techniques*) désignent les opérations permettant aux données à transmettre de circuler au sein d'un réseau NoC [154]. Ces opérations sont effectuées au niveau de chaque routeur traversé par un paquet depuis sa source jusqu'à la destination finale. Les données reçues et/ou renvoyées par chaque routeur sont sous forme de *phits*. Ces trames qui ont la même taille physique que le canal d'interconnexion entre deux routeurs, seront affectées à l'une des sorties du routeur selon l'adresse de la destination du paquet à transmettre.

Il existe deux techniques d'aiguillage généralement adoptées dans les réseaux sur puce : *Circuit Switching et Packet Switching*.

La technique d'aiguillage *Circuit switching* : Cette technique consiste à réserver un chemin entre la source et la destination, qui sera suivi pour envoyer tous les paquets d'un message de manière successive. Cette technique d'aiguillage est basée sur deux étapes. La première étape consiste à définir et réserver le chemin de transfert. Un *flit d'entête* (*header flit*) procède à l'initialisation de la communication. Son rôle est de réserver temporairement le chemin que le paquet va traverser pour arriver jusqu'à sa destination finale. Si cette étape s'est passée correctement (pas de collisions avec d'autres routeurs essayant de communiquer), un acquittement positif est rétro propagé pour confirmer la réservation du chemin. A ce moment, la deuxième étape commence. Au cours de cette deuxième étape, les paquets suivants seront envoyés successivement sur le chemin réservé. En cas de détection d'une collision, pendant la première étape, un acquittement négatif sera envoyé à la source pour relancer la procédure de communication [155, 156].

La technique *circuit switching* est caractérisée par sa large bande passante ainsi que par une latence de transmission faible au cours de la seconde étape. Par contre, elle présente quelques inconvénients du point de vue d'extensibilité. Entre autre, elle n'est pas adaptée pour des systèmes comportant un nombre de modules élevé, car le temps de réservation d'un circuit devient très important pouvant bloquer autres communications et influer sur les performances globales du système [157].

La technique d'aiguillage *Packet Switching* : Comme il est indiqué par son nom, la technique *Packet Switching* repose sur l'acheminement des paquets. Contrairement à la technique précédente, on n'a pas besoin de passer par une étape de réservation d'un chemin à suivre par tous les paquets d'un message. Chaque paquet établit son chemin au cours de la transmission d'un routeur à un autre. Chaque paquet est traité par le routeur se trouvant sur son chemin vers sa destination finale. Avec cette technique, il n'y a pas de circuits à réserver avant d'envoyer le paquet ce évitant de demander un temps supplémentaire d'établissement de chemin et bloquer temporairement le réseau. Par contre, la latence de transmission dans cette technique augmente à cause des encombrements au niveau des routeurs qui doivent fréquemment effectuer l'aiguillage de plusieurs trames à la fois [157]. Il existe trois types de technique *Packet switching* : *Store-And-Forward* (*SAF*), *Virtual-Cut-Trough* (*VCT*) et *Wormhole Switching* (*WS*). Pour la technique *SAF*, les données se propagent d'un routeur à

un autre sous forme de paquets, dès qu'un routeur est prêt à stocker toutes les trames d'un paquet [158]. Dans la technique *VCT*, qui est caractérisée par un temps de latence plus faible, les données sont traitées sous forme de *flits*. La transmission des paquets d'un routeur à un autre dépend de la capacité de stockage des routeurs en termes de *flits* de paquets. Dans ces deux techniques, l'inconvénient réside dans l'utilisation des *buffers* de grande taille. Par conséquent, l'implémentation matérielle est couteuse en termes de ressources matérielles nécessaires. La technique *WS* présente une solution plus légère en termes de ressources nécessaires. Dans cette technique d'aiguillage, la taille des *buffers* au niveau des entrées de chaque routeur peut être réduite à la taille du canal d'interconnexion. D'où, un *phit* stocké provisoirement dans un routeur attend que le routeur voisin sur son chemin vers sa destination finale soit capable de l'accepter pour qu'il soit transféré d'un routeur à l'autre. En divisant le message sur plusieurs *phits* circulant librement sur le réseau, les situations d'embouteillage apparaissent plus fréquemment par rapport aux techniques *SAF* et *VCT*.

Le choix de la technique d'aiguillage à adopter dépend des besoins en communication d'un système sur puce. Dans le cas où les messages entre deux modules du système sont composés de plusieurs paquets de grande taille, la technique d'aiguillage *circuit switching* est la plus adaptée. D'autre part, pour des échanges de messages de petite taille, la technique d'aiguillage *Packet Switching* de type *WS* se présente comme la meilleure approche à adopter.

iii) Algorithme de routage

L'algorithme de routage définit le chemin suivi par un message dans le réseau à partir de la source jusqu'à la destination finale. Les algorithmes de routage sont classés suivant différents critères : *statique ou dynamique*, *minimal ou non-minimal* et *distribué ou source* [154]. Dans les algorithmes de *routage statique* (*static routing*), le chemin de transfert d'un paquet d'une source à une destination est fixe et identique pour tous les paquets empruntant ce chemin lors d'une communication. Ce type d'algorithmes ne prend pas en compte l'état courant du réseau en termes de charge et d'occupation des canaux d'interconnexion. Par conséquent, le processus de routage est simple et ne demande pas de fonctions logiques complexes pour son implantation [159]. D'un autre côté dans les algorithmes de *routage dynamiques* (*dynamic routing*), le chemin de transfert d'un paquet entre deux nœuds (source et destination) est défini au cours de la transmission et s'établit en fonction des conditions du réseau. Donc, le circuit de communication est variable et les paquets suivent rarement le même chemin entre la source et la destination. Ces algorithmes sont plus coûteux en termes de ressources par rapport aux algorithmes statiques, notamment à cause de la logique d'acheminement étant plus complexe. De plus, ils proposent des avantages d'auto-adaptation pouvant améliorer les performances globales du système. Nous pouvons citer, parmi les algorithmes de *routage dynamique*, l'algorithme *fully-adaptative* [160], *odd-even* [161] et *congestion look-ahead* [162].

Selon un deuxième critère, les algorithmes de routage peuvent être classifiés en algorithmes

minimaux ou *non-minimaux*. Cette classification est en fonction de la distance parcourue par un paquet d'une source à une destination. Les algorithmes de *routage minimaux* sont caractérisés par l'établissement d'un chemin le plus court ou les chemins les plus courts s'ils en existent plusieurs. L'algorithme de routage *XY* est un exemple des algorithmes minimaux [159]. Par contre, la contrainte de chemin le plus court n'est pas prise en considération dans les algorithmes non-minimaux où le chemin est choisi selon la disponibilité du réseau. L'avantage de ces algorithmes est leur capacité de distribuer la charge de communication sur tout le réseau et donc d'éviter les situations d'encombrement. Ces algorithmes utilisent des logiques de routage plus complexes. Les algorithmes *dynamiques* sont souvent classés comme des algorithmes *non-minimaux*.

Les algorithmes de routage peuvent aussi être classifiés selon l'emplacement d'exécution de la décision du routage ainsi que l'emplacement de stockage de ces informations. On distingue les algorithmes de routage *source* et *distribué*. Dans ces derniers, le choix du circuit est effectué au niveau des routeurs. Ainsi, une table de routage est enregistrée au niveau de chaque routeur. En fonction de cette table, le routeur décide le chemin convenable pour le routage d'un paquet à une destination désignée. D'autre part, dans les algorithmes de routage sources, le chemin de routage est désigné par la source. Celle-ci ajoute les informations de routage à chaque paquet, permettant son acheminement jusqu'à'à la destination. Par conséquent, l'inconvénient de ce type de routage réside dans le nombre de bits réservés pour l'intégration des informations de routage.

iv) Topologies des réseaux sur puce

Une topologie définit la façon selon laquelle les nœuds sont distribués sur le réseau et surtout la structure d'interconnexion utilisée pour la liaison des routeurs. En littérature, plusieurs topologies des réseaux sur puce sont disponibles. Ces topologies peuvent être classées en trois grandes catégories [154] : *les réseaux directs*, *les réseaux indirects* et *les réseaux irréguliers*.

Les réseaux directs : Dans les réseaux directs, chaque nœud est composé d'un module (*PE, Mémoire, IP, bloc E/S*, etc. . .) relié à un routeur par l'intermédiaire d'un module d'interfaçage (*NI*). Les routeurs sont inter-connectés via des communications point à point, selon une structure de voisinage déterminée par la distribution topologique des nœuds sur le réseau. On peut distinguer différents types de topologie directe dont chacun est caractérisé par sa structure. La figure 3.5 illustre quelques types de réseaux sur puce à topologie directe. Les réseaux sur puce directs le plus souvent utilisés sont les réseaux *maillés en 2D*, *torus*, *folded torus* et *octagon* [149, 152, 163].

Ces topologies sont caractérisées par une large bande passante globale dont la taille dépend du nombre de nœuds sur le réseau. D'autre part, les performances de communication peuvent être améliorées avec une meilleure connectivité. Par contre, cela sera au détriment de la consommation d'énergie ainsi que du cout d'implantation en termes de ressources matérielles. Par conséquent, chaque choix représente un compromis entre la connectivité et les performances exigées, et le coût d'implantation.

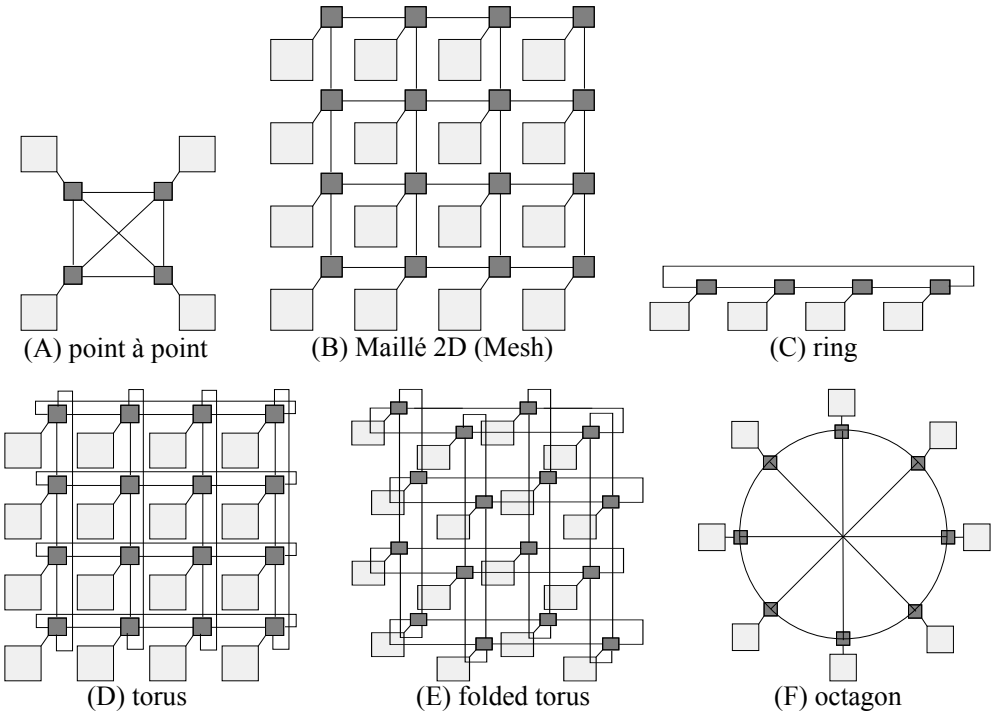


FIGURE *3.5 – Topologies des réseaux sur puce de type direct*

Les réseaux indirects : Dans les réseaux de type indirect, les modules du réseau sont connectés à un routeur par l'intermédiaire d'un *NI* comme dans les réseaux directs. Sauf que les routeurs, dans cette catégorie de topologie, peuvent être reliés à plusieurs modules ou même connectés entre eux sans être connectés à des modules fonctionnels. Un exemple de réseaux sur puce de type indirect est une structure de plusieurs unités de calcul *PE* inter-connectées via un seul routeur. Ce type de réseau nommé *« Crossbar »* est simple de mise en oeuvre. D'un autre côté, son manque d'extensibilité et son coût relativement important en termes de connectivité proportionnelle au nombre de *PE* font que ce type de réseau est rarement utilisé.

La figure 3.6 présente deux autres exemples de topologie indirecte. Dans la topologie de type *fat tree*, les routeurs sont interconnectés dans une *structure hiérarchique* sous forme d'*arbre* où la racine est composée d'un seul routeur et les modules sont connectés aux extrémités nommées « *feuilles* ». Avec cette topologie, le principe similaire au réseau crossbar est utilisé avec la différence au niveau du routage qui est assuré par plusieurs routeurs disposés à des niveaux hiérarchiques différents. Cette topologie est caractérisée par une bande passante réduite par rapport aux réseaux de type direct ainsi qu'une surcharge hétérogène au niveau des routeurs qui croit en passant d'un niveau hiérarchique inférieur vers un autre plus proche de la racine. La deuxième topologie présentée la figure 3.6(b) est une variante de la structure *fat tree*. La *topologie papillon* ou « *Butterfly* » permet d'améliorer la bande passante du réseau tout en offrant une meilleure connectivité par rapport au réseau *« Crossbar »*. Les topologies de type indirect sont moins fréquentes pour l'implémentation des réseaux sur puce, bien que des exemples de ce type de réseau existent en

littérature [150, 151, 164] .

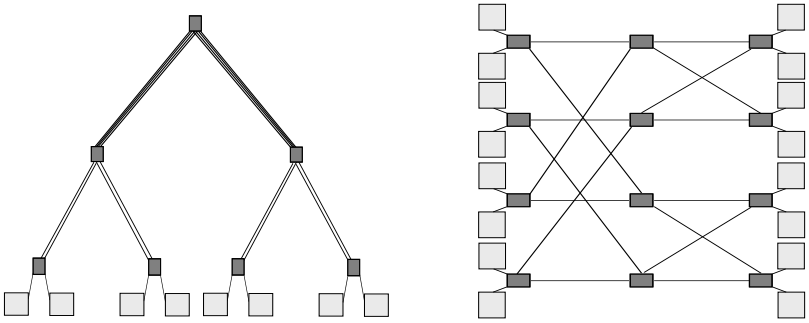


FIGURE *3.6 – Topologie de réseaux sur puce de type indirect*

La *topologie 2D maillée (2D mesh)* semble être la topologie la plus fréquente dans le domaine des réseaux sur puce [12]. Ceci est principalement dû à la structure d'interconnexion de cette topologie qui est régulière et permet ainsi d'adopter les algorithmes de routage plus simples à mettre en œuvre, sans oublier la capacité d'extensibilité offerte par cette structure qui croît linéairement avec le nombre de nœuds.

3.2.5 Approche de communication pour une architecture matérielle de la carte SOM extensible et flexible

L'architecture d'une carte *SOM* est également une architecture distribuée d'éléments de calcul appelés neurones, où tous ces neurones doivent non seulement effecteur des calculs en parallèle mais également échanger des données entre eux lors de chaque itération dans les phases d'apprentissage et/ou de rappel. Une carte *SOM* est un système dont les éléments constitutifs sont très connectés et ont des besoins très forts en communication : d'une part, pour recevoir les données d'entrée liées à l'environnement externe et d'autre part pour échanger les distances calculées lors des phases d'apprentissage et de rappel. La figure 3.7(a) présente l'architecture matérielle classique de la carte auto-organisatrice. Elle est composée de trois couches : *couche d'entrée*, *couche compétitive* et *couche de sorties.* Tous les éléments de vecteur d'entrée sont massivement reliés aux neurones de la couche compétitive via des liens physiques directs. Ces éléments sont traités en parallèle sur tous les neurones de la couche compétitive. Ensuite, les résultats de calcul sont envoyés au *module de comparaison global* sur la couche de sorties pour élire le neurone gagnant. Cette architecture permet d'optimiser les performances de la carte auto-organisatrice avec des communications à délai réduit, un traitement parallèle et une comparaison parallèle centralisée. Par contre, le changement des paramètres de la carte (taille, dimension du vecteur d'entrée, etc) dans la majorité des cas ne peut pas être effectué sur ces architectures dites classiques et demande en contrepartie des efforts de conception considérables pour reformuler les mécanismes d'échange de données entre les neurones d'une part, et adapter le mécanisme de comparaison des résultats obtenus (distances euclidiennes calculées) entre tous les neurones d'autre part.

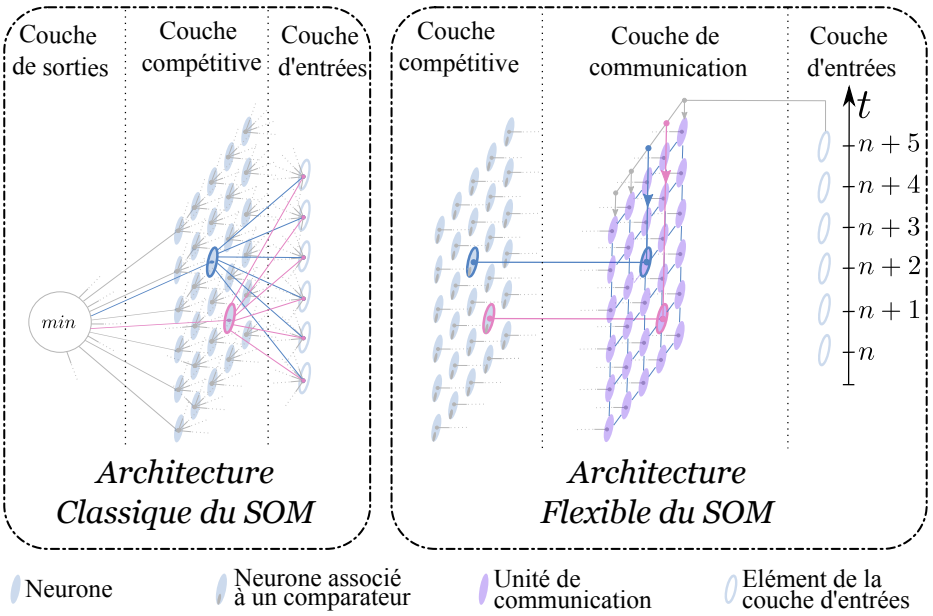


FIGURE 3.7 – *Comparaison de l'architecture proposée avec l'architecture classique de la carte SOM*

Dans nos travaux, nous visons l'adaptabilité des implémentations matérielles menant vers plus de flexibilité, tout en gardant les avantages des architectures matérielles qui sont essentiellement les performances et la consommation d'énergie réduite. Pour atteindre cet objectif, des changements radicaux doivent être effectués au niveau de l'architecture matérielle classique. Ces changements peuvent être présentés sur trois niveaux :

— Au niveau d'un neurone où le traitement effectué doit pouvoir être reconfigurable à la volée (online) et pourra s'adapter à la dimension des vecteurs d'entrée sans changements de l'architecture (aucun effort de conception supplémentaire à fournir).

— Au niveau de la fonction de comparaison qui jusqu'à présent dans la majorité des architectures matérielles s'effectue de façon globale limitant son adaptabilité à la taille de la couche compétitive.

— Au niveau des mécanismes d'échange d'informations entre les neurones, qui doivent être capables de supporter aussi bien les changements des paramètres de la carte auto-organisatrice que les interconnexions entre les neurones.

Par conséquent, une architecture matérielle de la carte auto-organisatrice devra être composée d'une couche d'entrée dont les éléments sont distribués et connectés à tous les neurones. De plus, le comparateur global doit également être distribué sur des modules de comparaison locaux couplés avec les neurones sur la couche compétitive. Enfin, toutes les communications devront être gérées par une couche de communication indépendante. Avec une telle approche, toutes dépendances architecturales de l'algorithme de *Kohonen* aux changements paramètres principaux peuvent être évitées. Cette architecture est présentée la figure 3.7(b).

Un réseau sur puce de part ses propriétés pourra se prêter bien à ces besoins en communication d'une architecture matérielle de la carte *SOM* évoqués précédemment. On peut facilement remarquer la ressemblance entre une topologie d'un *NoC* et la topologie d'une carte *SOM*. Un

réseau sur puce de type *direct* avec une *topologie 2D mesh* semble être le système de communication le plus adapté aux implémentations matérielles d'une carte *SOM* flexible dont le principe est présenté figure 3.7(b). Les deux types de données circulant au sein d'une carte *SOM*, les vecteurs d'entrée ou des stimuli souvent composés de plusieurs éléments peuvent être acheminés et échangés vers ou entre neurones en utilisant la technique d'aiguillage la plus adaptée à ce type de communication étant le technique *Circuit Switching* et présentée dans le section précédente. Par contre, les informations échangées entre les neurones, notamment les distances calculées avec les identifiants des neurones correspondants, sont souvent de petite taille ne dépassant pas dans leurs formes un ou deux *phits*, d'où la technique la plus adaptée dans ce type d'échange sera la technique *Packet switching* de type *WS*. De plus, l'extensibilité inhérente d'un réseau sur puce apportera à l'architecture matérielle de la carte *SOM* utilisant un réseau sur puce comme moyen de communication cette possibilité de pouvoir élargir à la volée le nombre de neurones nécessaires au système et aux besoins applicatifs à un moment donné.

Dans la section suivante, nous proposons une architecture matérielle de la carte *SOM* basée sur une approche de communication de type réseau sur puce. Tous les détails architecturaux permettant d'adapter l'algorithme de Kohonen sur une telle architecture seront présentés. De plus, l'architecture présentée est également évaluée sur un exemple de quantification de couleurs dans la dernière partie et également comparée avec les architectures matérielles classiques de la carte *SOM* en termes de performances et ressources matérielles nécessaires pour son implantation sur une technologie reconfigurable de type *FPGA*.

3.3 Architecture matérielle extensible et flexible de la carte auto-organisatrice

Afin de trouver un compromis entre la flexibilité proposée par les implémentations logicielles et les performances obtenues avec des implémentations matérielles, nous proposons *une architecture matérielle adaptable de la carte auto-organisatrice de Kohonen*. La flexibilité offerte par cette approche consiste en la capacité de changer dynamiquement la structure de la carte en termes de nombre de neurones sur la couche compétitive ainsi qu'en termes de dimension de vecteur de poids de chaque neurone. L'architecture proposée peut être facilement reconfigurée pour changer la topologie de voisinage ainsi que les paramètres d'apprentissage. Par conséquent, cette approche d'amélioration de la flexibilité profite des performances des implémentations matérielles avec une architecture parallèle de type *Multiple Instruction Multiple Data* (*MIMD*) où les neurones sur lesquels les éléments de vecteur d'entrée et de poids sont traités d'une manière sérielle en pipeline, avec une propagation systolique pendant la phase de compétition.

Pour garantir une implémentation matérielle performante avec un coût d'implémentation relativement réduit, quelques simplifications, généralement utilisées dans les implémentations matérielles classiques, ont été proposées.

3.3.1 Simplifications algorithmiques proposées

Le calcul de la distance dans l'algorithme *Self-Organising Map* (*SOM*) est une opération de base exécutée sur tous les neurones de la carte. Dans la version originale de l'algorithme proposée par *Kohonen*, cette opération est le calcul de la distance *Euclidienne* comme présenté avec l'équation 3.1. Bien évidemment, la valeur de la distance calculée est considérée comme une valeur positive ($D_{L2} > 0$). Par conséquent, pour deux neurones N_1 et N_2, si $D_{L2_{N_1}} \leq D_{L2_{N_2}}$ alors $D^2_{L2_{N_1}} \leq D^2_{L2_{N_2}}$. Pour cette raison, l'utilisation du carré de la distance *Euclidienne* D^2_{L2} est privilégiée par rapport à la distance D_{L2}, permettant d'une part d'aboutir au même résultat dans la procédure de la recherche de la distance minimale, et ainsi l'identification du neurone gagnant, et d'autre part de simplifier les calculs effectués au sein d'un neurone [55, 108]. Par conséquent, la complexité globale de l'algorithme de Kohonen est diminuée et son implémentation matérielle est simplifiée. De surcroît, le temps de calcul de la distance sera également réduit par le temps nécessaire pour effectuer une opération de *racine carrée*. D'autre part, la suppression de cette opération permet d'omettre, au niveau de chaque neurone, l'opérateur de racine carrée relativement complexe permettant ainsi de réduire également le nombre des ressources matérielles utilisées. Dans notre architecture, la distance utilisée est la distance D^2_{L2}. Cette distance est calculée suivant l'équation 3.2.

$$D_{L2} = \sqrt{\sum_{i=1}^{D} (\xi_i - \mu_i)^2} \tag{3.1}$$

$$D^2_{L2} = \sum_{i=1}^{D} (\xi_i - \mu_i)^2 \tag{3.2}$$

avec, ξ_i et μ_i, étant respectivement les éléments du vecteur d'entrée et les éléments du vecteur de poids, et *D* étant la dimension des deux vecteurs.

D'autre part, le calcul réalisé selon les équations 3.3 et 3.4 pour l'adaptation des éléments de vecteur de poids au niveau de chaque neurone, est d'une grande complexité pour les implémentations matérielles. Cette complexité réside dans *la fonction de voisinage* $h_{c,i,j}(t)$ présentée par l'équation 3.4, avec *c* étant l'indice du neurone gagnant et *(i,j) ∈ ([1,L], [1,K])* étant la position du neurone dans la carte *SOM* de dimension $\mathrm{L} \times \mathrm{K}$. Dans sa forme originale, cette équation utilise plusieurs opérateurs arithmétiques complexes.

$$\overrightarrow{\mathrm{W_i}}(t+1) = \overrightarrow{\mathrm{W_i}}(t) + h_{C,i}(t) \times (\overrightarrow{\mathrm{X}} - \overrightarrow{\mathrm{W_i}}) \tag{3.3}$$

$$h_{C,i,j}(t) = \boldsymbol{\alpha}(t) \times \exp(-\frac{\left\| \overrightarrow{\mathrm{r_c}} - \overrightarrow{\mathrm{r_{i,j}}} \right\|}{2\sigma^2(t)}) \tag{3.4}$$

Afin de réduire les ressources matérielles utilisées et d'améliorer les performances temporelles de l'exécution de cette opération, une approche de simplification a été proposée en litté-

rature [55, 72, 126, 165]. Cette simplification consiste à remplacer *la fonction gaussienne*, telle qu'elle est présentée avec l'équation 3.4, par une simple opération de décalage de $2^{-S_{c,i,j}}$ bits calculée selon l'équation 3.5 comme suit :

$$h_{c,i,j}(t) = \frac{1}{2^{S_{c,i,j}(t)}} \tag{3.5}$$

Le nombre de bits à décaler $S_{i,j}(t)$ est calculé en fonction de deux paramètres, comme présenté avec l'équation 3.7 : la position du neurone par rapport au neurone gagnant $R_{c,i,j}$ (voir équation 3.6) et l'évolution de la phase d'apprentissage $\beta(t)$. Ce dernier paramètre s'incrémente suivant une règle empirique proposée par Kohonen dans ses travaux initiaux sur l'algorithme, après $L \times K \times 30$ itérations d'apprentissage (avec $L \times K$ le nombre de neurones sur la couche compétitive) afin d'éviter le sur-apprentissage [55].

$$R_{c,i,j} = \left\| \overrightarrow{r_c} - \overrightarrow{r_{i,j}} \right\| \tag{3.6}$$

$$S_{c,i,j}(t) = R_{c,i,j} + \beta(t) \tag{3.7}$$

Ainsi, l'équation 3.3 est remplacée par l'équation 3.9. Dans cette équation, les vecteurs de poids des neurones appartenant au cercle de voisinage du neurone gagnant de rayon $R_v(t)$ seront rapprochés au vecteur d'entrée avec un taux de voisinage et un taux d'apprentissage traduits par une simple fonction de décalage. De point de vue d'implantation matérielle, la mise en œuvre de cette opération est réalisée par une addition précédée par une simple fonction de décalage des valeurs de différence $\overrightarrow{\Delta}(t)$ entre l'élément du vecteur d'entrée $\overrightarrow{X}$ et celui du vecteur de poids $\overrightarrow{W}(t)$. D'autre part, pour réduire le temps de mise à jour des poids, une mémoire a été réservée pour l'enregistrement des valeurs de différence $\overrightarrow{\Delta}(t)$. Cette différence est calculée selon l'équation 3.8, au cours de l'opération de calcul de la distance. Afin d'éviter la répétition de ce calcul nécessaire non seulement dans la phase de compétition mais également dans la phase d'adaptation, la mémoire souvent utilisée pour l'enregistrement des vecteurs d'entrée $\overrightarrow{X}$ dès leur réception dans les architectures matérielles classiques, sera dédiée dans l'architecture proposée à l'enregistrement des valeurs $\overrightarrow{\Delta}(t)$ avant de calculer le carré de la distance utilisée. Ensuite, ce vecteur sera réutilisé pendant la mise à jour des vecteurs de poids $\overrightarrow{W}(t)$, comme présenté dans l'équation 3.9.

$$\overrightarrow{\Delta}(t) = \overrightarrow{X} - \overrightarrow{W}(t) = \{(\xi_1 - \mu_1(t)), (\xi_2 - \mu_2(t)), \ldots, (\xi_D - \mu_D(t))\} \tag{3.8}$$

$$\overrightarrow{W}(t+1) = \begin{cases} \overrightarrow{W}(t) + \frac{\overrightarrow{\Delta}(t)}{2^{S_{c,i,j}(t)}} & si\ R_{c,i,j} \leq R_v(t) \\ \overrightarrow{W}(t) & sinon \end{cases} \tag{3.9}$$

Les simplifications adoptées ont pour objectif de réduire la complexité matérielle de l'architecture ainsi que le temps d'exécution de l'algorithme. Donc, par cela nous avons cherché

à simplifier l'implémentation matérielle de la carte auto-organisatrice. Le deuxième objectif de notre approche est d'ajouter de la *flexibilité* à l'implémentation matérielle de ce modèle neuronal.

3.3.2 Principe de la propagation systolique dans l'architecture proposée

L'une des approches de distribution de l'opération d'élection du neurone gagnant dans les architectures matérielles de la carte *SOM* est l'approche systolique [55, 139]. Cette approche consiste en utilisation de plusieurs comparateurs locaux avec *une propagation de proche en proche ou dite systolique* de la distance minimale [55, 139]. L'approche systolique de la carte auto-organisatrice utilisée dans notre architecture est présentée en figure 3.8. Dans l'architecture proposée, les neurones sont distribués sur des étages nommés étages systoliques. Pour une carte *SOM* de taille $L \times K$ on peut distinguer $N_{es} = L + K - 1$ *étages* systoliques.

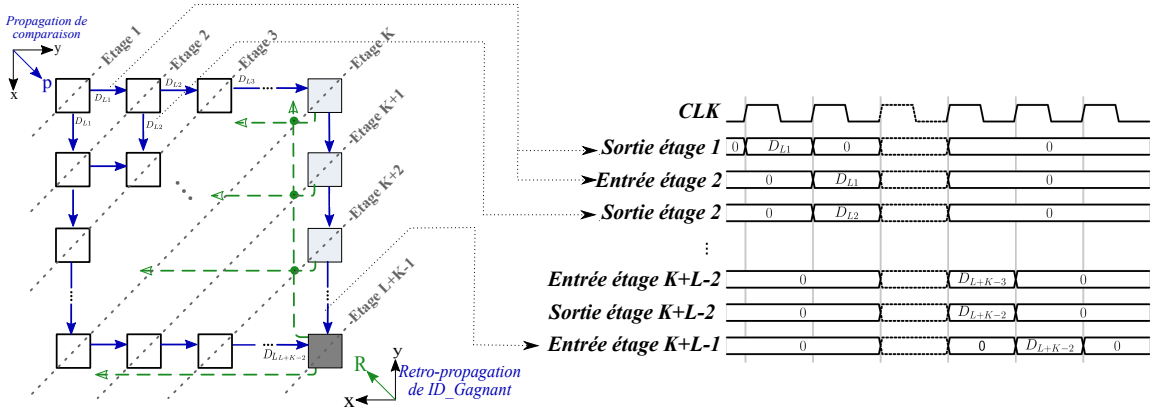


FIGURE *3.8 – Architecture systolique de la carte* SOM *et les chronogrammes de propagation systolique utilisée dans la recherche du neurone gagnant (indice indique l'étage systolique)*

La propagation systolique utilisée dans la recherche du neurone gagnant pour un vecteur d'entrée commence à partir de l'étage *1* composé d'un seul neurone se trouvant à l'intersection de la ligne *1* et la colonne *1*. Ce neurone envoie sa distance calculée vers ses voisins adjacents (à droite et en bas) situés sur l'étage systolique d'indice *2*. Ces derniers effectueront une comparaison entre leurs distances et la distance envoyée par le neurone situé à l'étage systolique 1. Ensuite, la distance minimale locale de l'étage systolique d'indice 2 sera envoyée avec l'identité du neurone correspondant vers les voisins adjacents de l'étage systolique d'indice *3*. De cette façon, la comparaison des distances calculées par tous les neurones s'effectue au niveau de chaque étage systolique et se propage jusqu'à l'étage $K + L - 1$ au niveau duquel l'identité du neurone gagnant sera connue. A ce moment, cette identité sera *rétro-propagée* vers tous les neurones de la couche compétitive pour lancer la phase d'adaptation.

De cette façon, la comparaison globale est divisée sur $L \times K - 1$ comparaisons locales. Chaque comparaison est effectuée sur un neurone associé (i, j) selon sa position sur la grille $L \times K$ suivant l'équation 3.10.

$$c_{i,j} = \begin{cases} \underset{(l,k)\in W_1}{argmin}\ D^2_{L2,(l,k)}, & si\ i = 0, 0 < j < K \\ \underset{(l,k)\in W_2}{argmin}\ D^2_{L2,(l,k)}, & si\ 0 < i < L, j = 0 \\ \underset{(l,k)\in W_3}{argmin}\ D^2_{L2,(l,k)}, & si\ 0 < i < L, 0 < j < K \end{cases} \tag{3.10}$$

avec W_i désignant l'ensemble des neurones participant à la comparaison dont les identités sont calculées comme suit :

$$\begin{aligned} W_1 = &\quad \{(i,j),(i,j-1)\}, && si\ i = 0, 0 < j < K \\ W_2 = &\quad \{(i-1,j),(i,j)\}, && si\ 0 < i < L, j = 0 \\ W_3 = &\quad \{(i-1,j),(i,j),(i,j-1)\}, && si\ 0 < i < L, 0 < j < K \end{aligned} \tag{3.11}$$

Enfin, la distance minimale ainsi que l'identité du neurone gagnant seront identifiées après $L + K - 2$ *opérations de comparaison*. Au cours d'un cycle de comparaison, tous les neurones appartenant à l'étage systolique *i* effectueront leurs comparaisons locales en parallèle. Ensuite, les résultats de comparaison obtenus au cours de ce cycle seront utilisés sur l'étage *i+1* pendant le cycle suivant. Par conséquent, le parallélisme de l'opération d'élection du neurone gagnant sera réduit par rapport à celui de l'architecture classique dont le facteur d'accélération A_{Hw} par rapport à une exécution séquentielle est présenté avec l'équation 3.12, pour avoir un facteur d'accélération $\mathrm{A_{sys}}$ présenté avec l'équation 3.13.

$$A_{Hw} = L \times K \tag{3.12}$$

$$A_{sys} = \frac{L \times K}{L + K - 2} \tag{3.13}$$

avec $L \times K$ étant nombres de lignes et colonnes sur la grille de la couche compétitive.

Bien que l'utilisation d'une approche systolique réduise les performances d'exécution de l'opération d'élection du neurone gagnant, cette approche permet d'offrir à l'architecture matérielle de la carte *SOM* proposée la propriété d'extensibilité lui permettant un changement dynamique de nombre de neurones sur la couche compétitive. Cette extensibilité est offerte en introduisant des communications homogènes entre les neurones appartenant à des étages systoliques voisins en les rendant indépendantes de la taille de la carte *SOM* utilisée. Les communications entre les neurones de la carte *SOM* relatives à *la propagation systolique* au cours de la phase de compétition sont présentées dans le tableau 3.2, tandis que les communications utilisées pour *la rétro-propagation* sont résumées dans le tableau 3.3. Nous pouvons constater que, quelle que soit la phase de calcul (compétition ou rétro propagation), un neurone appartenant à un étage systolique d'indice i aura à communiquer uniquement avec les neurones voisins les plus proches

appartenant à des étages systoliques adjacents i – 1 et i + 1. Si à chaque neurone appartenant à un étage systolique est offerte la possibilité de paramétrage des neurones auxquels il envoie les données propres à une phase de calcul, l'extensibilité de la carte *SOM* n'aura plus de limite. Nous verrons dans la section suivante que dans l'architecture proposée, cette possibilité de configuration est offerte notamment grâce à l'utilisation de l'approche de communication de type réseau sur puce, dont les détails propres à l'architecture proposée sont également présentés dans la section suivante.

TABLEAU *3.2 – Communication de propagation systolique au cours de la phase de compétition*

Position *(i,j)*	**Source**	**Destination**	**Messages**
$1 \leq i \leq L-1,\ 1 \leq j \leq K-1$	(i,j)	$(i+1,j)\ et\ (i,j+1)$	$D^2_{L2,C_{i,j}}\ et\ C_{i,j}$
$1 \leq i \leq L-1,\ j = K$	(i,j)	$(i+1,j)$	$D^2_{L2,C_{i,j}}\ et\ C_{i,j}$
$i = L,\ 1 \leq j \leq K-1$	(i,j)	$(i,j+1)$	$D^2_{L2,C_{i,j}}\ et\ C_{i,j}$

TABLEAU *3.3 – Communication de rétro-propagation au cours de la phase d'adaptation*

Position *(i,j)*	**Source**	**Destination**	**Message**
$1 \leq i \leq L-1,\ j = K$	(L,K)	(i,K)	C
$1 \leq i \leq L,\ 1 \leq j \leq K-1$	(i,K)	(i,j)	C

3.3.3 Communication à base de NoC

Les opérations de l'algorithme de Kohonen peuvent être réparties en deux catégories : *les opérations de communication* et *les opération de traitement*. Les opérations de communication consistent en la distribution des vecteurs de poids vers tous les neurones de la carte *SOM*, en propagation systolique de la comparaison des distances minimales locales et en la diffusion de l'identité du neurone gagnant vers tous les neurones. Par ailleurs, le calcul de la distance et l'adaptation des vecteurs de poids sont classées comme des opérations de traitement. Dans l'architecture matérielle proposée de la carte *SOM*, pour réduire au maximum la dépendance architecturale de la structure de la carte auto-organisatrice avec ses propres paramètres, l'exécution des opérations évoquées est distribuée sur des modules paramétrables indépendants dont le comportement dépend des données d'entrée et paramètres reçus de manière dynamique. La figure 3.9 présente l'architecture principale de la carte *SOM* proposée dans ces travaux de recherche et nommée l'architecture *Self-Organizing Map Based on Network-on-Chip* (*SOM-NoC*).

L'architecture proposée est divisée en deux couches, comme présenté en figure 3.9. La *couche de traitement* chargée des opérations de traitement est composée d'un ensemble de modules de traitement distribués sur une grille à deux dimensions $L \times K$. Sur cette couche, les modules ne

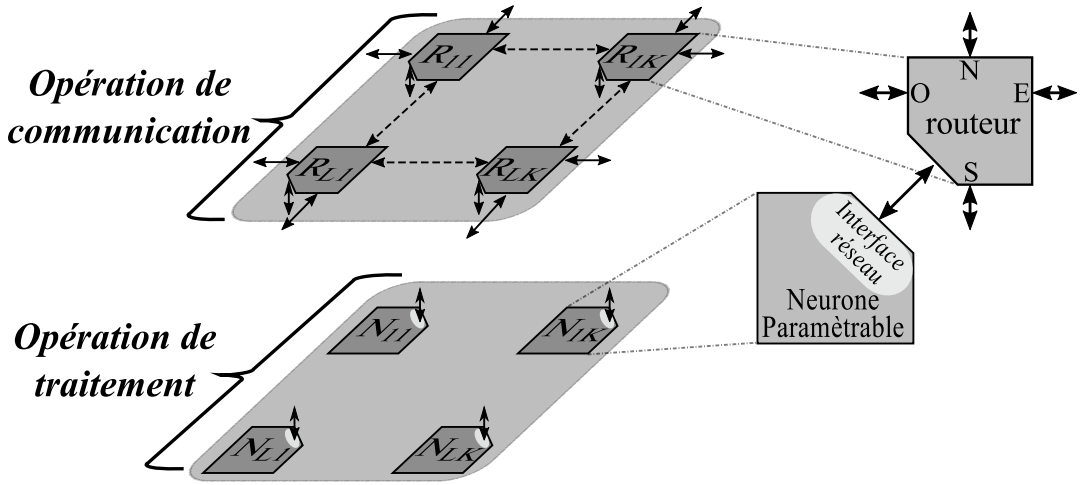


FIGURE *3.9 – Architecture SOM-NoC à deux couches : couche de traitement et couche de communication*

sont pas connectés entre eux ou inter-connectés. A la réception d'un message sur la connexion unique du module allant vers la couche de traitement, une interface réseau « *NI* » associée interprète le message reçu et ordonne l'exécution de l'opération de traitement décodée au module, représentant l'unité de traitement. Par conséquent, étant donné l'architecture parallèle des modules de traitement, l'exécution des opérations peut être effectuée en parallèle sur tous les modules de la couche. D'autre part, l'échange de données entre modules de traitement est effectué via la *couche de communication* par le biais des messages circulant entre les modules de communication de cette couche. En effet, la couche de communication est composée d'un ensemble de modules de communication distribués de la même façon que les modules de la couche de traitement. De cette façon, chaque module de communication est associé à un module de traitement via une connexion physique bidirectionnelle uniforme. De plus, pour assurer la communication entre les différents modules de l'architecture, les modules de la couche de communication sont interconnectés entre eux selon une *topologie 2D maillée*, comme présenté en figure 3.9. Le dialogue entre deux modules de l'architecture est initié par les *NI* correspondantes et assuré par le réseau de routeurs. Dans l'architecture proposée, un neurone de la carte auto-organisatrice est représenté par un couple de modules de traitement et de communication. Chaque neurone est identifié par un couple d'indice calculé selon sa position sur la grille. Ces indices peuvent être utilisés comme identité du neurone pour l'exécution de l'algorithme de Kohonen ou comme adresse pour adresser un message à un neurone au cours d'une opération de communication.

L'architecture du réseau sur puce utilisé dans notre approche est présentée en figure 3.10(a). Cette architecture *2D maillée* est composée des *routeurs* inter-connectés entre eux associés à des *modules de traitement PE*. Le dialogue entre un module *PE* et son *routeur* associé est assuré par une *interface réseau* (*NI*) dont la fonction principale est de coder sous forme d'un *paquet* (avant l'envoi) et de décoder le *paquet* (après réception) les données échangées entre différents *PE*.

La figure 3.10(d) illustre la structure des paquets circulant dans le réseau en utilisant la tech-

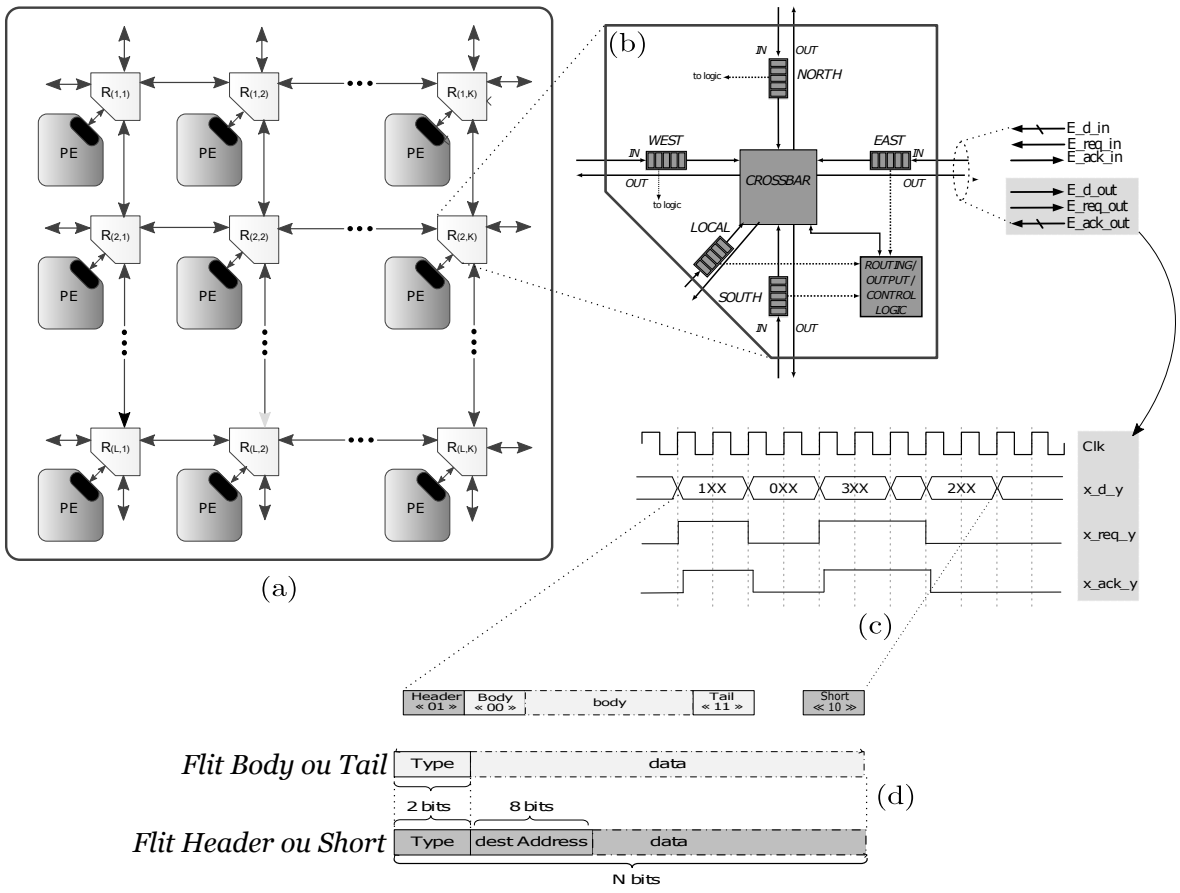


FIGURE *3.10 – (a) Architecture 2D maillée du NoC utilisé dans cette architecture matérielle de la carte SOM, (b) Architecture interne d'un routeur, (c) Protocole d'échange d'information utilisé entre deux routeurs (Alternate Bit Protocol), (d) Structure des messages circulant dans le NoC utilisé*

nique de commutation *Wormhole Switching* (*WS*). Un message peut être présenté sur plusieurs paquets appelés *flits* (ensemble de bits pouvant être présentés simultanément sur le support physique). Avec cette technique de commutation, on trouve trois types de *flit* : *Header, Body* et *Tail*. Le type d'un *flit* est codé sur *ses deux premiers bits*. Le *flit Header*, identifié par le code « *01* », ouvre la communication et "*réserve*" le circuit de communication aux *flits* suivants. Ce *flit* contient l'adresse de la destination. Le flit *Body*, identifié par le code « *00* », ne contient que les données à transmettre entre les *PE* sur le circuit précédemment réservé. Le *flit Tail*, identifié par le code « *11* », a la même structure que *body*. Ce *flit* permet de terminer la communication et de libérer les routeurs pour un nouveau message. D'autre part, les messages courts, qui peuvent être présentés sur un seul paquet, sont codés sur un *flit* de type *Short*. Ce *flit*, identifié par le code « *10* », contient l'adresse de la destination. Il calcule son circuit de communication au cours de la transmission et ne réserve aucun routeur lors de son passage.

Le module principal du *NoC* utilisé est le routeur dont l'architecture est présentée figure 3.10(b). Ce module assure la transmission des paquets entrants vers la sortie convenable jusqu'à ce qu'ils atteignent la destination finale. Un *routeur* est composé d'un « *Crossbar* » dont le rôle est d'effectuer une opération d'aiguillage pour établir différentes connexions filaires entre les

liaisons d'entrée et de sortie du routeur selon l'*algorithme de routage* et la politique d'ordonnancement adoptés. *L'arbitrage des paquets* dans le routeur ainsi que *la fonction d'aiguillage* sont gérés avec un *bloc logique de contrôle*. Par exemple, si plusieurs paquets arrivent simultanément sur différentes entrées d'un routeur et demandent la même sortie vers leurs destinations finales, ce bloc de contrôle doit arbitrer et déterminer la priorité de transfert de ces paquets. Dans ce cas, les paquets en attente doivent être stockés dans des mémoires tampons jusqu'à ce que leur transfert soit planifié. C'est la raison principale pour laquelle chaque routeur possède des mémoires tampons ou des « *Buffer* » d'entrée et/ou de sortie, dont le rôle est d'accepter temporairement les paquets avant leur transmission au PE local ou à un des routeurs voisins. Les liens d'interconnexion (en entrée et en sortie) d'un routeur isolé sont également illustrés en figure 2.5(b). L'interconnexion entre un routeur et ses routeurs voisins ou son *PE* associé est effectuée à base de six liens physiques (3 pour l'émission et 3 pour la réception) : un bus de données, un signal de requête de transfert et un signal d'acquittement de réception respectivement notés par *_d_, _req_ et _ack_*.

L'échange de données entre deux modules (*routeur/routeur ou routeur/PE*) est effectué à base du protocole de communication « *Alternate Bit Protocol* (*ABP*) ». La figure 3.10(c) présente les chrono-grammes des signaux illustrant les échanges de données entre deux modules à base de ce protocole. Un *flit* est positionné sur le bus *x_d_y* (avec $x \in (N, E, S, W\ ou\ L)$ et $y \in (in\ ou\ out)$). Ce *flit* est accompagné par un changement d'état sur le lien *x_req_y* associé au bus de données. A la réception de cette requête, le module receveur procède à l'aiguillage du *flit* reçu vers la sortie convenable selon sa destination finale. Si cette sortie est libre, l'état du signal sur le lien *x_ack_y* sera modifié. Si non, la communication sera bloquée et le *flit* sera stocké dans la mémoire tampon en attendant la libération de la sortie souhaitée. Dans ce cas, le *flit* suivant ne sera envoyé que lorsqu'on aura un changement d'état sur le lien d'acquittement.

En utilisant cette approche de communication à base de réseau sur puce, nous avons réussi à séparer les opérations de communication de celles de traitement au sein d'une carte de Kohonen. Par conséquent, la dépendance de l'architecture proposée à la structure et aux paramètres de la carte auto-organisatrice est fortement réduite. D'autre part, l'utilisation de ce type de *NoC*, à base d'une technique d'aiguillage mixte et d'un algorithme de routage dynamique, nous a permis d'offrir de la flexibilité et d'adaptabilité au niveau des opérations de communication, la communication représentant une véritable limitation en termes d'adaptabilité dans les architectures matérielles classiques de la carte auto-organisatrice.

Par ailleurs, la nouvelle approche à base de réseau NoC sur laquelle l'architecture proposée est essentiellement basée ne présente aucune limite au niveau de la taille de la couche compétitive à utiliser à un moment donné. Autrement dit, l'utilisation de la technique de communication à base des réseaux sur puce permet d'ajouter la propriété d'extensibilité et d'adaptabilité à l'architecture proposée que nous l'avons appelée l'architecture *Self-Organizing Map Based on Network-on-Chip* (*SOM-NoC*).

3.3.4 Extensibilité de l'architecture matérielle SOM-NoC

En plus de la flexibilité, l'originalité majeure de l'architecture matérielle *SOM-NoC* proposée est son extensibilité. Cette propriété est illustrée à la figure 3.11 sur un exemple d'extension d'une architecture 2×2 vers une architecture 2×3 d'une carte auto-organisatrice à base de l'architecture *SOM-NoC*. Cette extension est effectuée par l'ajout d'une carte *SOM-NoC* plus petite de taille 2×1, sans passer par la phase de conception.

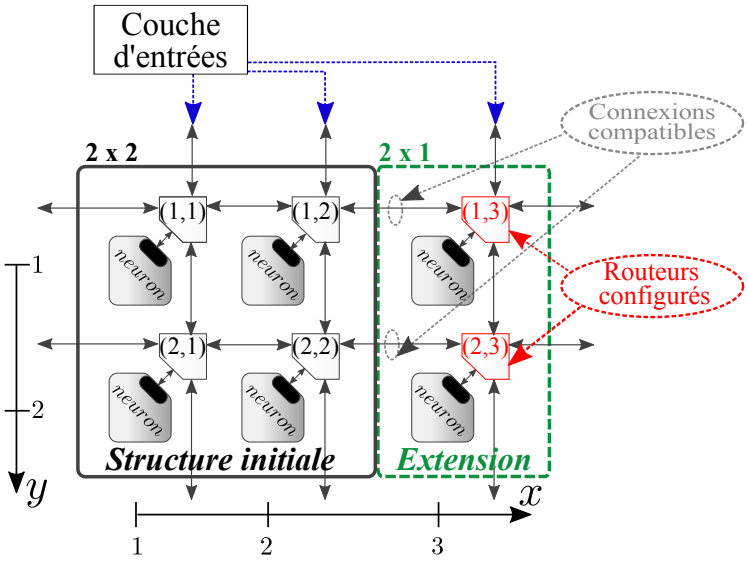


FIGURE *3.11 – Extensibilité de l'architecture matérielle SOM-NoC de la carte de Kohonen*

Dans la structure initiale on remarque l'existence de différents liens de communication uniformes sur les extrémités de l'architecture. Ces liens peuvent être utilisés pour connecter des nouveaux modules à l'architecture initiale. Dans l'exemple présenté, les liens physiques de communication situés à l'extrémité haute de l'architecture sont utilisés pour connecter la couche d'entrée provenant de l'environnement externe à l'architecture *SOM-NoC* (*liens en bleu*). L'extension de la structure initiale de l'architecture *SOM-NoC* de taille 2×1 est présentée dans le cadre vert et est positionnée à l'extrémité droite de la structure initiale. Cet ajout physique de nouveaux neurones/routeurs doit respecter la règle de compatibilité de liens physiques : les mêmes liens physiques que ceux utilisés dans la structure initiale doivent être utilisés pour connecter ces nouveaux éléments. Une fois physiquement rajoutée, chaque nouvelle extension doit être proprement configurée pour pouvoir faire partie de la nouvelle structure de la carte de Kohonen. Cette configuration consiste à paramétrer les couples routeurs/neurones nouvellement rajoutés dans la nouvelle structure plus grande de façon à ce que leurs adresses ainsi que l'algorithme de routage pour les routeurs uniquement, correspondent à leur nouvelle position sur la grille. A savoir que chaque neurone/routeur doit avoir un identifiant unique dans la nouvelle structure, pour garantir le bon déroulement de leurs opérations dans toutes les phases d'exécution de l'algorithme de Kohonen. De cette façon, tous les modules de la nouvelle structure de l'architecture *SOM-NoC* pourront communiquer après la phase de configuration. Ceci permettra aux modules de traite-

ment reconfigurables (neurones) de cette architecture de s'adapter de manière dynamique aux nouveaux paramètres de la carte auto-organisatrice via des messages de configuration envoyés par le réseau *NoC*.

3.4 Architecture interne d'un neurone de l'architecture SOM-NoC

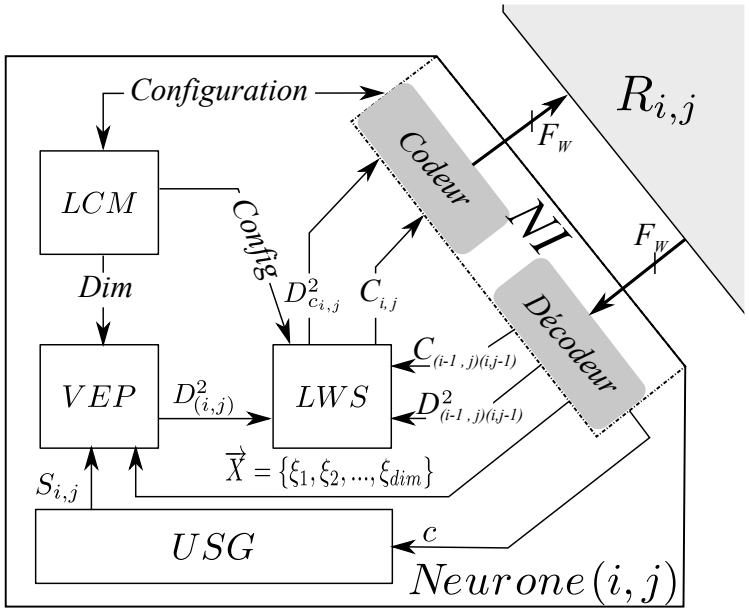


FIGURE *3.12 – Architecture d'un neurone de l'architecture SOM-NoC*

Comme nous l'avons cité précédemment, l'exécution de l'*algorithme de Kohonen* dans l'architecture proposée est assurée par des modules distribués effectuant localement une partie du traitement global. Pour exploiter au mieux la flexibilité et l'extensibilité offerte par l'approche proposée au niveau global, il est indispensable de proposer une adaptabilité au niveau local, c'est à dire au niveau des modules distribués étant les neurones. Chaque neurone doit être configurable afin de pouvoir adapter son comportement et ainsi le(s) traitement(s) à effectuer aux paramètres de la carte auto-organisatrice demandés à un instant donné ainsi que sa position et son identité sur la couche compétitive. Dans ce cadre, nous proposons une architecture reconfigurable d'un neurone. Le schéma synoptique de l'architecture interne d'un neurone est présenté à la figure 3.12. Cette architecture est composée de cinq modules répartis en deux groupes. Dans le premier groupe, on distingue trois *modules de traitement* : le processeur d'éléments de vecteur ou *Vector Element Processor* (*VEP*), le module de mise à jour des poids ou *Update Signal Generator* (*USG*) et le comparateur de distance local ou *Local Winner Search* (*LWS*) ; dans le second groupe on distingue un *module de communication* ou *NI* et un *module de configuration locale* ou *Local Configuration Management* (*LCM*).

3.4.1 Processeur d'éléments de vecteur - *Vector Element Processor (VEP)*

Le module *VEP* assure les traitements à effectuer sur les éléments de vecteurs de poids et d'entrée au niveau d'un neurone. On distingue trois types de traitement : *calcul de la distance*, *adaptation des poids* et *récupération des poids*. Comme présenté à la figure 3.13, le processeur VEP est un chemin de donnée composé d'un certain nombre d'opérateurs arithmétiques séparés et synchronisés par des registres lui permettant un traitement *en pipeline*. Dans le neurone présenté, les éléments des vecteurs de poids et d'entrée sont analysés d'une façon sérielle. Un *générateur d'indice* d'éléments est implémenté pour gérer l'adressage des mémoires contenant ces vecteurs. Ce générateur d'indice est paramétrable selon la dimension des vecteurs d'entrée et de poids étant spécifique à l'application.

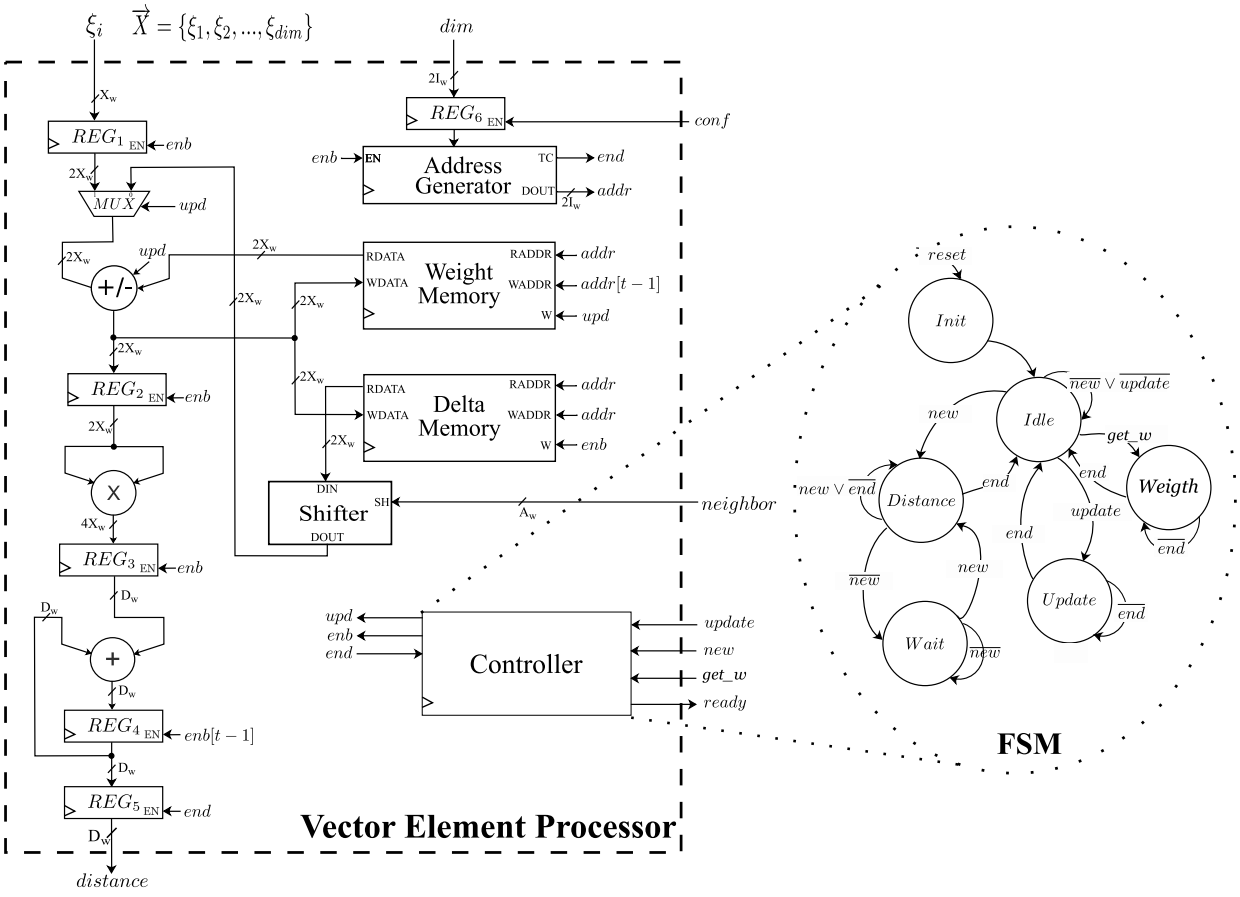


FIGURE *3.13 – Architecture du processeur d'éléments de vecteur - Vector Element Processor (VEP)*

L'architecture du processeur *VEP* contient deux mémoires : *une mémoire des poids* dans laquelle les éléments de vecteur des poids sont stockés et *une mémoire des deltas* dans laquelle les différences δ_i extraites pendant le calcul de la distance sont enregistrées afin d'être réutilisées au cours de l'adaptation. De cette façon, les performances de traitement au sein d'un neurone sont optimisées. De plus, pour optimiser les ressources matérielles utilisées, certains opérateurs arithmétiques sont réutilisés dans différentes opérations. La gestion du processeur VEP est effectuée par un contrôleur réalisé sous forme d'une machine à états finis dont le diagramme d'états est

présenté à la figure 3.13. Dans ce contrôleur, on distingue 6 états :

— ***Init*** – Initialisation des poids du neurone.

— ***Idle*** - Un état de repos (attente des ordres pour effectuer une opération).

— ***Distance*** – Calcul de la distance dès la réception d'un élément du vecteur d'entrée.

— ***Wait*** – Attente des éléments du vecteur d'entrée en cas de retard de réception.

— ***Update*** – Mise à jour du vecteur de poids à la réception de l'identité du neurone gagnant.

— ***Weigth*** – Récupération des éléments du vecteur de poids à la demande.

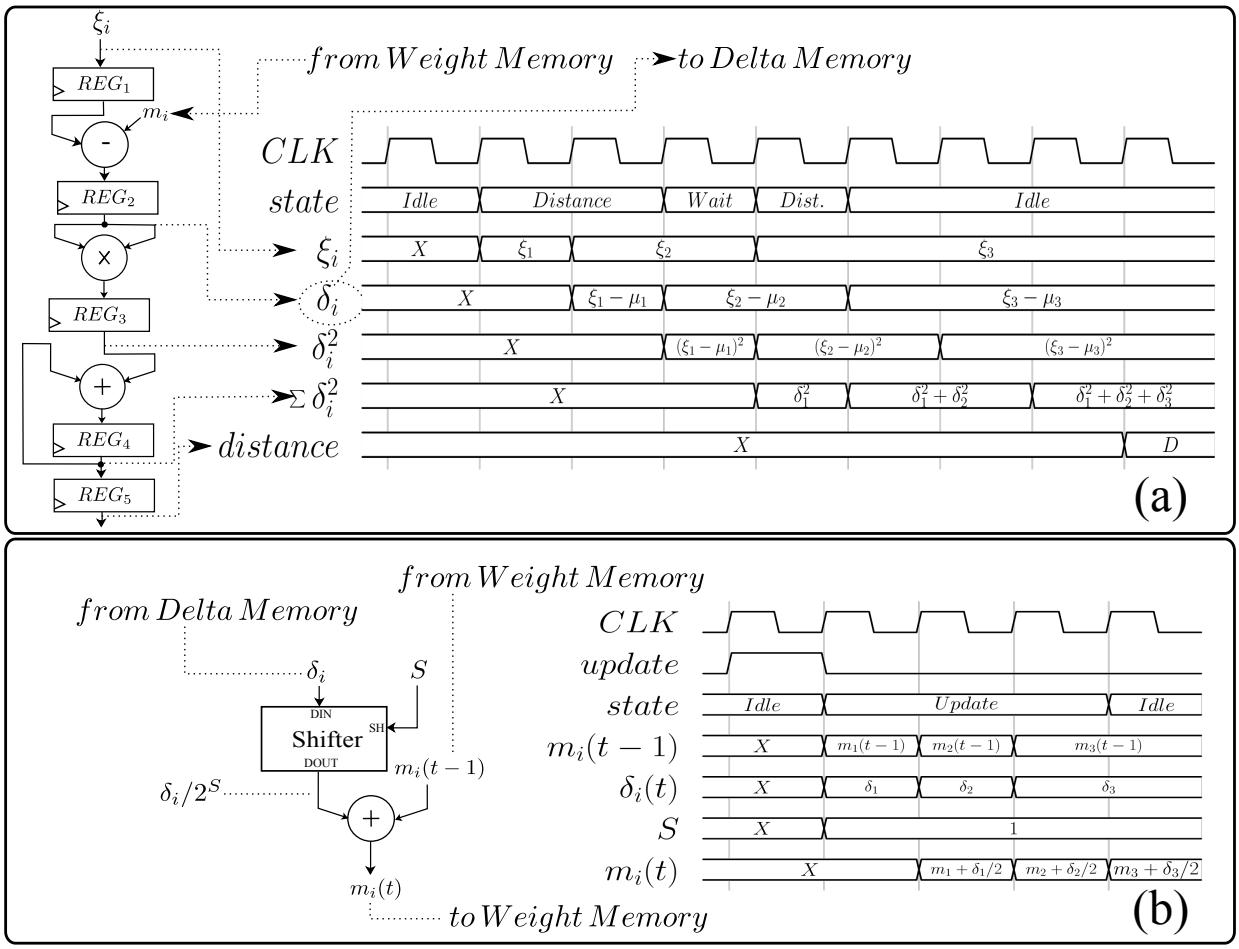


FIGURE *3.14 – (a) Circuit de calcul de la distance ; (b) Circuit d'adaptation des poids*

Pour simplifier la compréhension du fonctionnement de l'architecture du neurone présentée figure 3.13, les opérations de calcul de la distance et de l'adaptation sont présentées séparément respectivement sur les figures 3.14(a) et (b). D'un autre côté, l'opération de récupération des poids consiste en une simple opération de lecture dans la mémoire des poids. Le circuit de calcul de la distance assure le calcul du carré de la distance *Euclidienne* suivant l'équation 3.2. Ce circuit est composé d'une série de soustracteur, multiplieur et additionneur synchronisés par des registres et alimentés par les données externes ainsi que les données récupérées de la mémoire des poids. Après le traitement de tous les éléments des vecteurs d'entrée, le résultat final de cette opération est récupéré dans le registre REG_5 de l'architecture (voir figure 3.14). De plus, les différences entre les éléments du vecteur de poids et ceux du vecteur d'entrée sont enregistrées

dans la mémoire des deltas au cours de traitement de chaque élément. Ces valeurs sont ensuite réutilisées dans l'opération d'adaptation suivant l'équation 3.9. Cette opération est assurée par un registre à décalage qui effectue un décalage de *S bits* sur la différence récupérée de la mémoire des deltas selon l'équation 3.5. Finalement, ce résultat sera additionné avec l'ancienne valeur du vecteur de poids et le résultat est sauvegardé à sa place dans la mémoire des poids. Ainsi, la mise à jour du vecteur de poids du neurone est effectuée.

3.4.2 Module de mise à jour des poids - *Update Signal Generate (USG)*

Le module de mise à jour des poids *USG* est responsable de la génération des signaux de mise à jour au cours de la phase d'adaptation. A la réception de l'identité du neurone gagnant $C_{i,j}$, le module *USG* calcule sa position par rapport à ce neurone gagnant. Selon cette position ainsi que le taux d'apprentissage identifié par le paramètre β, le nombre *S* représentant le nombre de décalage à effectuer sur la valeur de δ sera calculé suivant l'équation 3.7. Le circuit assurant cette opération est présenté à la figure 3.15.

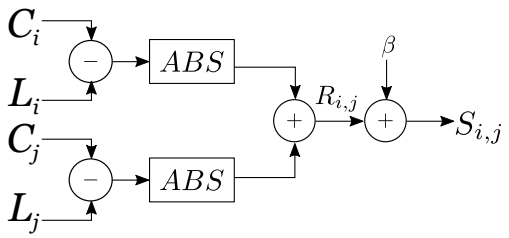


FIGURE *3.15 – Circuit de calcul des paramètres de la mise à jour*

Après avoir vérifié l'appartenance du neurone au rayon de voisinage $R(t)$, le nombre de décalage à effectuer sera envoyé vers le module *VEP*, accompagné d'un signal de lancement de mise à jour nommé *Update*. Enfin, les paramètres d'apprentissage seront modifiés selon le déroulement de l'apprentissage, en incrémentant le *taux d'apprentissage* et en décrémentant le *rayon de voisinage*.

3.4.3 Comparateur de distance local - *Local Winner Search (LWS)*

Le module *LWS* est un comparateur local associé à chaque neurone. Son rôle est de trouver la distance minimale entre la distance locale calculée par le processeur *VEP* et celles envoyées par les neurones voisins directs se trouvant en amont du sens principal de *la propagation systolique* choisie. Ensuite, le résultat de cette comparaison est envoyé vers les neurones voisins les plus proches se trouvant en aval du sens de la propagation systolique. Ces neurones voisins sont déterminés et configurés dans la phase de configuration du système, avant de démarrer tout traitement. Les données nécessaires pour la comparaison (distance locale, distances minimales des voisins et identités des gagnants locaux des voisins) sont définies selon la configuration locale du neurone,

déterminant les identités des neurones locaux desquels les distances doivent parvenir avant d'effectuer cette comparaison. Dans la phase de compétition, après le calcul de la distance locale par le neurone et la réception des résultats de comparaison des voisins directs, ces données et leurs identités correspondantes sont préparées et mises à disposition de ce comparateur *LWS*.

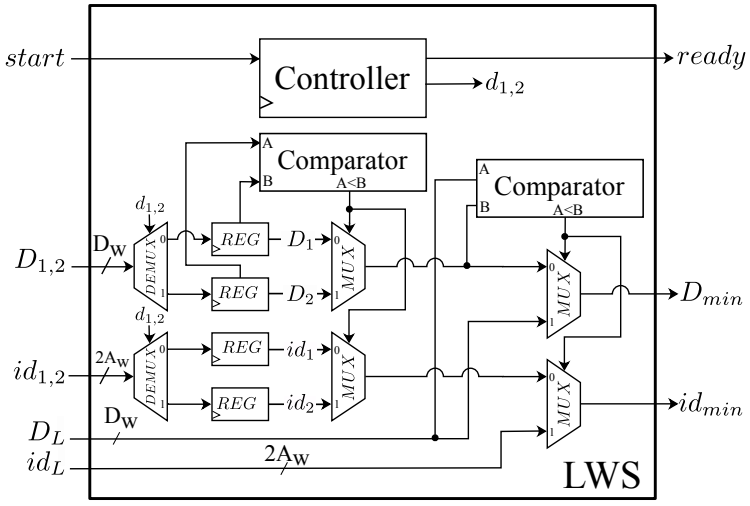


FIGURE *3.16 – Architecture du comparateur local LWS*

L'architecture du comparateur *LWS* est présentée à la figure 3.16. On remarque que la comparaison s'effectue à base d'un circuit asynchrone. Après la préparation des données de comparaison, la comparaison est effectuée suivant l'équation 3.10. Ensuite, un signal sera envoyé vers l'interface réseau pour signaler la fin de cette opération et permettre de continuer *la propagation systolique* des résultats de comparaison au niveau global.

3.4.4 Interface réseau - *Network Interface (NI)*

Les communications dans l'architecture proposée sont effectuées à base du réseau *NoC* comme présenté dans la section 3.3.3. Pour assurer le dialogue entre le *routeur* et son neurone associé, ce dernier est équipé d'une *NI* dont le rôle consiste à traduire les messages des communications à envoyer ou reçus par le neurone. L'interface réseau est composée d'un codeur et un décodeur comme présenté à la figure 3.12.

Les données générées par le neurone, et qui seront envoyées via le *NoC*, sont collectées par le codeur qui assure également leur transmission. Ce dernier les prépare sous une forme compréhensible par les routeurs ainsi que par les décodeurs des neurones destinataires. Cette opération, appelée opération de paquetage, consiste en la mise en paquet des messages. Ces paquets ont la structure présentée à la figure 3.10(d). Par conséquent, le codeur ajoute à chaque paquet le *type de flit*, *l'adresse de destinataire*, et le *type des données* envoyées. En effet, dans la zone réservée aux données dans un *flit*, *4 bits* sont utilisés pour coder le type de l'information envoyée. Le tableau 3.4 résume les différents types d'information circulant dans le *NoC* dont chacun est identifié par un code différent.

TABLEAU *3.4 – Différents types des messages utilisés par l'architecture SOM-NoC*

Code	Type	Destination	Description
« *0000* »	Configuration	*LCM*	Nombre de neurones
« *0001* »	Configuration	*LCM*	Dimension des vecteurs
« *0010* »	Élément du vecteur d'entrée	*VEP*	Premier élément du vecteur d'entrée (pour l'apprentissage)
« *0011* »	Élément du vecteur d'entrée	*VEP*	Premier élément du vecteur d'entrée (pour le rappel)
« *0100* »	Élément du vecteur d'entrée	*VEP*	Le reste des éléments du vecteur d'entrée
« *0101* »	Compétition	*LWS*	La distance minimale locale
« *0110* »	Compétition	*LWS*	L'identité du neurone gagnant local
« *0111* »	Adaptation	*USG/Output*	L'identité du neurone gagnant global
« *1000* »	Sortie	*Output*	La distance minimale globale
« *1001* »	Sortie	*Output*	Premier élément du vecteur de poids
« *1010* »	Sortie	*Output*	Le reste des éléments du vecteur de poids

D'autre part, les messages envoyés vers le neurone sont reçus par le décodeur appelé également récepteur. Ce dernier décode les flits reçus pour en extraire les informations essentielles du message. Cette opération est appelée opération de dépaquetage. Ensuite, le décodeur envoie aux modules concernés les instructions ainsi que les données nécessaires relatives aux instructions reçues. Ces instructions générées par le décodeur assurent une synchronisation et bon déroulement des différentes opérations de l'algorithme *SOM* sur les différents modules d'un neurone. A part les différents types de messages listés dans le tableau 3.4, on trouve dans l'architecture *SOM-NoC* des messages de configuration permettant d'assurer l'adaptation du comportement de chaque neurone aux paramètres et à la structure de la carte auto-organisatrice. Ce type de message est particulièrement destiné au module de configuration locale *LCM*, abordé dans la sous-section suivante.

3.4.5 Module de configuration locale - *Local Configuration Module (LCM)*

Chaque neurone dans l'architecture proposée est autonome. Il participe à l'exécution de l'algorithme *SOM* en assurant un ensemble d'opération à base de son *architecture dynamiquement reconfigurable*. Sa position dans la couche compétitive détermine son comportement effectif à un moment donné. Les changements des paramètres et de la structure souhaités de la carte auto-organisatrice de manière dynamique sous-entend que chaque neurone reçoit un message de (re-)configuration, qui est destiné au module *LCM*. Les informations contenues dans ce message sont la taille de la couche compétitive en nombre de neurones et la dimension des vecteurs d'entrée et

de poids.

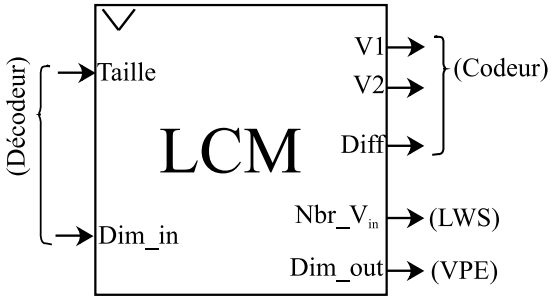


FIGURE *3.17 – Architecture du module de configuration locale LCM*

La figure 3.17 présente le schéma bloc du module LCM avec ses principales entrées et sorties. La taille de la couche compétitive en nombre de neurones et la position du neurone dans cette couche compétitive servent de base au module *LCM* pour calculer les principaux paramètres qu'il doit fournir à l'interface *NI* assurant l'envoi et la réception des données. En particulier, le module *LCM* définit à partir des données globales (taille de la carte, dimension de vecteur) les voisins auxquels son neurone parent doit envoyer les informations de compétition (distance minimale locale et l'identité du neurone qui a calculé cette distance) suivant la propagation systolique, ainsi que le nombre d'informations qu'il doit recevoir avant de désigner un neurone gagnant local. Ces valeurs fournies par le module *LCM* sont envoyées respectivement vers le transmetteur et le comparateur local. De plus, le module *LCM* doit également définir le rôle principal de son neurone parent dans la phase d'adaptation. En effet, au cours de la phase d'adaptation, la diffusion de l'identité du neurone gagnant est assurée uniquement par quelques neurones de l'architecture globale. Les neurones concernés par cette opération s'auto-identifient à l'aide du module *LCM* qui selon la taille de la couche compétitive et leur position dans la couche compétitive configure leurs transmetteurs. Le tableau 3.5 résume les différentes configurations du transmetteur ou décodeur qui garantissent le bon fonctionnement de l'algorithme au cours de la phase de compétition et d'adaptation en fonction des coordonnées (i,j) de chaque neurone dans la carte auto-organisatrice. Nous pouvons constater qu'un neurone en fonction de sa position peut avoir jusqu'à deux neurones voisins directs auxquels les résultats de comparaison locale leur seront envoyés. A titre d'exemple, le neurone se trouvant à la position (L,K) où L et K sont les dimensions de la carte, n'a aucun neurone dans sa liste des neurones destinataires auxquels les résultats de comparaison devront être envoyés, puisque c'est lui qui découvre en premier l'identité du neurone gagnant global et lance la phase d'adaptation. De surcroît, le module *LCM* assure également l'envoi de la dimension des vecteurs d'entrée et de poids vers le processeur *VEP* pour lui indiquer le nombre d'éléments qu'il doit traiter d'une manière sérielle avant de fournir la distance.

TABLEAU *3.5 – Configuration du transmetteur (décodeur) selon la position (i,j) du neurone sur une structure $L \times K$*

Position	**Destination au cours de la propagation systolique**	**Diffusion au cours de la rétro-propagation**
$1 \leq i \leq L-1$, $1 \leq j \leq K-1$	$(i+1,j)$ $(i,j+1)$	*Null*
$i = L$, $1 \leq j \leq K-1$	$(i,j+1)$	*Null*
$1 \leq i \leq L-1$, $j = K$	$(i+1,j)$	*Par colonne*
$i = L$, $j = K$	*Rétro-propagation*	*Par ligne et colonne*

3.5 Architecture de la carte SOM adaptable à base de SOM-NoC

L'approche proposée précédemment permet d'ajouter des propriétés de flexibilité et d'extensibilité à une implémentation matérielle de la carte *SOM*. Dans cette section, nous proposons une architecture adaptable de la carte auto-organisatrice en se basant sur l'approche *SOM-NoC* proposée.

Pour avoir une architecture adaptable de la carte auto-organisatrice, nous avons rajouté à l'architecture *SOM-NoC* présentée précédemment un module de configuration globale. Cette architecture adaptable de la carte auto-organisatrice est présentée à la figure 3.18. Elle est composée d'une grille de neurones dynamiquement reconfigurables (*SOM-NoC*), d'un circuit de gestion des entrées/sorties nommé « *Input/Output Management Circuit* (*IOMC*) » connecté directement aux routeurs placés aux extrémités *Nord (haut)* et *Ouest (gauche)* du réseau *NoC* et d'un *module de gestion de la configuration globale* nommé « *Global Configuration Management* (*GCM*) ». Ce module est responsable de la configuration globale du système à son démarrage et au cours de son fonctionnement.

Une configuration consiste à fixer les paramètres et la structure de la carte *SOM* à utiliser dans le cadre d'exécution d'une application spécifique. Pour chaque nouvelle application, une nouvelle configuration doit être fixée en modifiant principalement la taille de la la couche compétitive ainsi que la dimension des vecteurs d'entrée et de poids. Ces paramètres sont ensuite envoyés par le circuit *IOMC* via le *NoC* vers tous les neurones actifs dans la nouvelle structure de la carte *SOM*.

A base de l'architecture proposée, l'implémentation matérielle de la carte auto-organisatrice peut s'adapter à des applications diverses dont les besoins structurels ainsi que les paramètres sont différents. A la réception d'une nouvelle configuration, chaque neurone procède à une reconfiguration locale au niveau du module *LCM*, qui comme expliqué précédemment prend en compte les nouveaux paramètres globaux pour définir les fonctions et le rôle du neurone parent au niveau local. La diffusion d'une configuration via le *NoC* vers tous les neurones de la carte nécessite une durée dépendant du nombre de routeurs à traverser depuis le module IOMC. Par conséquent, la structure du réseau doit être prise en considération pour calculer le temps de configuration total

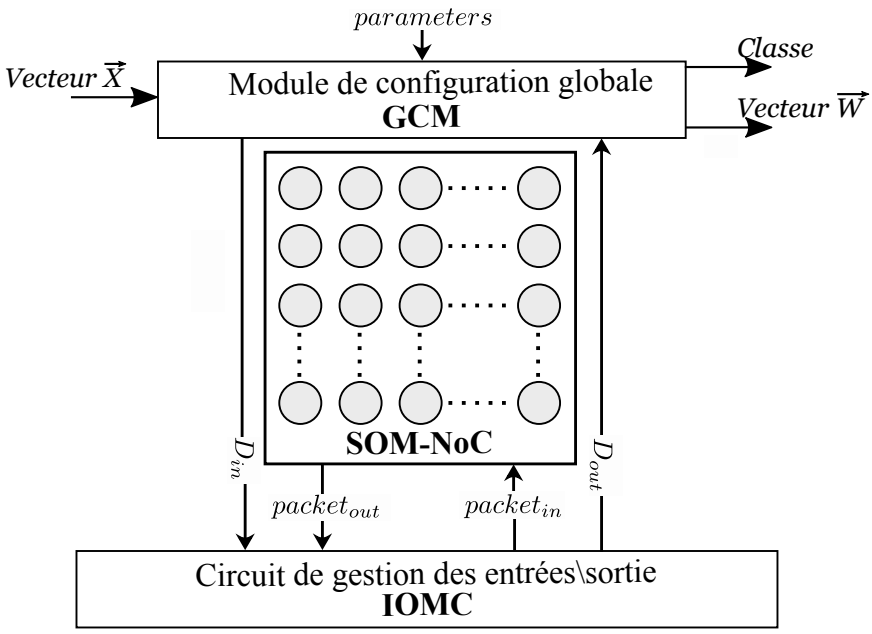


FIGURE *3.18 – Architecture d'une carte SOM adaptable à base de l'approche SOM-NoC*

nécessaire pour configurer tous les neurones (et routeurs en cas d'extensibilité) de l'architecture.

Du point de vue matériel, la structure du réseau implanté est fixée au cours de la phase de conception. Il s'agit bien d'une structure comportant un nombre maximal de neurones et routeurs pouvant être implantés sur le support physique choisi. La couche d'entrée située au niveau du module *GCM*, comme illustré à la figure 3.18, est réalisée sous forme d'un bloc mémoire composé de D^* *cases* dans lesquelles les éléments de vecteur d'entrée sont enregistrés avant d'être distribués vers tous les neurones de la couche compétitive. Par conséquent, la dimension maximale du vecteur d'entrée est définie dans la phase de conception et correspond à la taille du bloc mémoire D^*. D'autre part, le nombre maximal de neurones implantés $L^* \times K^*$ est aussi fixé au cours de la phase de conception. Ce nombre peut évoluer grâce à la propriété d'extensibilité de l'architecture *SOM-NoC* présentée plus tôt dans ce chapitre. Par conséquent, sur un support matériel physique défini à l'avance, le nombre maximal de neurones/routeurs est égal à $L^* \times K^*$ tandis que *la dimension maximale* D^* de vecteurs d'entrée est égale à la valeur correspondant à la taille du bloc mémoire utilisé dans le module *GCM*.

3.5.1 Procédure d'adaptation de l'architecture proposée

Du point de vue fonctionnel, le comportement de l'architecture proposée est adaptable aux paramètres et à la structure de la carte *SOM* utilisée à un moment donné. Cela est assuré non seulement par le module de configuration globale *GCM* mais également par la reconfiguration des neurones appartenant à la structure utilisée qui adaptent leurs comportements à ces nouveaux paramètres. La figure 3.19 présente un exemple d'adaptation dynamique de l'architecture proposée.

Dans l'exemple présenté figure 3.19, la carte auto-organisatrice est initialement composée de

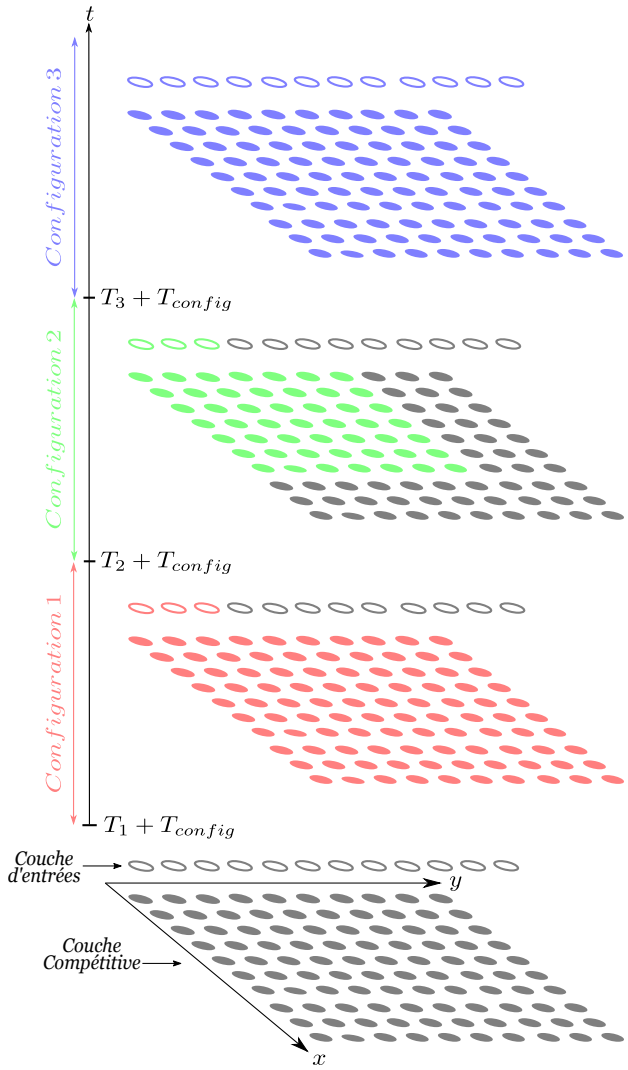


FIGURE *3.19 – Exemple d'adaptation dynamique de l'architecture proposée*

10×10 *neurones inactifs* sur la couche compétitive et *12 éléments de vecteur* sur la couche d'entrée. A l'instant T_1, le premier jeu de paramètres est envoyé vers les neurones. A la réception de ces nouveaux paramètres, les neurones inactifs se configurent à la structure composée de 10×10 *neurones* utilisant des vecteurs d'entrée de *dimension 3*. Après une période de configuration égale à T_{config} comme illustré à la figure 3.19, les neurones de la carte configurés à ces nouveaux paramètres sont prêts pour démarrer la phase d'apprentissage. A l'instant T_2, un nouveau jeu de paramètres est envoyé à l'architecture : le nombre de neurones demandé est plus petit que ceux qui sont actifs au moment de la demande : 7×7. A ce moment, une nouvelle configuration est effectuée en envoyant les nouveaux paramètres vers le module *GCM*, qui se charge de les envoyer à son tour vers les neurones désignés à participer à la nouvelle structure de la carte *SOM*. Ces neurones effectuent une nouvelle reconfiguration pour adapter leurs comportements aux nouveaux paramètres souhaités de la carte *SOM*. Avec cette configuration, seulement les neurones de couleur verte, comme illustré à la figure 3.19, sont activés et pourront par la suite traiter les vecteurs d'entrée dont la taille reste inchangée dans ce cas de figure. Les autres neurones de l'ar-

chitecture (colorés en gris, voir la figure 3.19) restent inactifs jusqu'à la réception des paramètres de la *configuration 3* à l'instant T_3, où l'étape de reconfiguration comme expliqué précédemment dans les deux cas de figure précédents recommence pour ce nouveau jeu de paramètres.

Avec cette approche, la configuration de l'architecture proposée pour l'adapter à une nouvelle application peut s'effectuer dynamiquement à n'importe quel instant. Dans le cas d'une reconfiguration, les données de la carte auto-organisatrice correspondant au jeu de paramètres et à la structure précédente seront réinitialisées et une nouvelle structure sera prête à être utilisée sans avoir besoin de passer par une nouvelle phase de conception.

3.5.2 Résultats d'implantation de l'architecture adaptable proposée

L'architecture proposée a été décrite en langage *VHSIC Hardware Description Language* (*VHDL*) et a été synthétisée pour une implantation sur une carte *Field-Programmable Gate Array* (*FPGA*) de type *VC707 Virtex-7* en utilisant l'outil *ISE Design Suite* de *Xilinx*. La fréquence maximale d'horloge obtenue avec cette implantation est de *250 MHz*. Cette fréquence n'est pas ou très peu affectée par les changements de taille de la couche compétitive de la carte auto-organisatrice ni par la dimension du vecteur d'entrée. Cela revient, d'une part à la distribution du traitement sur les modules élémentaires, et d'autre part, au traitement sériel des éléments des vecteurs d'entrée et de poids.

Les ressources matérielles nécessaires pour l'implantation physique de l'architecture adaptable dépendent du nombre de neurones sur la couche compétitive. Le tableau 3.6 résume les résultats d'implantation des modules de l'architecture sur une technologie FPGA Virtex-7 de Xilinx. Les résultats sont présentés en termes de nombre de registres, de LUT et de blocs DSP utilisés.

3.5.3 Performances de l'architecture proposée

Les paramètres utilisés pour l'évaluation des performances de l'architecture proposée sont affichés dans le tableau 3.7. Il s'agit respectivement des paramètres spécifiques au neurone (X_w, D_w, D, T_c, T_d), au routeur (A_w, F_w, T_r) et à la structure globale ($L \times K, D, N_{es}$).

Pour généraliser l'évaluation des performances temporelles de l'architecture proposée, le temps d'exécution de chaque étape de l'algorithme de Kohonen est quantifié par une formule

TABLEAU *3.6 – Ressources matérielles utilisées pour l'implantation sur une carte FPGA VC707 Virtex-7*

Ressources	**GCM**	**IOMC**	**Neurone**	
			PE	**Routeur**
Nombre de registre	*8 354*	*1 264*	*8 562*	*1 005*
Nombre de LUT	*6 235*	*1 683*	*6 313*	*1 335*
Nombre de DSP	*1*	*0*	*1*	*0*

TABLEAU *3.7 – Paramètres de l'architecture proposée*

Paramètre	**Valeur**	**Description**
X_w	*8 Bits*	Précision d'un élément de vecteur
A_w	*4 Bits*	Précision de l'adresse
D_w	*24 Bits*	Précision de la distance
F_w	*38 Bits*	Précision d'un flit
$L \times K$	$2^{A_w} \times 2^{A_w} = 16 \times 16$ *Neurones*	Taille de la couche compétitive
D	$2^{2 \times A_w} = 256$ *Elements*	Dimension des vecteurs d'entrée
N_{es}	$L + K - 1 = 31$ *Etages*	Nombre d'étages systoliques
T_r	*2 clk*	Délai de passage par routeur
T_c	*1 clk*	Temps de paquetage
T_d	*1 clk*	Temps de dépaquetage

dépendant des paramètres de la carte utilisée. Le tableau 3.8 présente ces expressions temporelles ainsi que les résultats obtenus dans le cas d'une carte auto-organisatrice dont les paramètres sont affichés dans le tableau 3.7. Les valeurs affichées dans la dernière colonne tiennent compte de la fréquence d'horloge obtenue dans la phase d'implantation (250MHz).

TABLEAU *3.8 – Performances de l'architecture proposée*

Etape	**Formule** (clk)	**Temps** (ns)
Calcul de la distance	$T_{cd} = D + 4$	*1 040*
Comparaison locale	$T_{sw} = T_c + 3 \cdot T_r + 2 \cdot T_d$	*36*
Propagation systolique	$T_p = T_{sw} \times (N_{sl} - 1)$	*1 080*
Rétro-propagation d'identifiant du neurone gagnant	$T_{rp} = 2 \cdot T_r + (T_c + 2 \cdot T_d) \times (N_{sl} - 1)$	*376*
Mise à jour des poids	$T_{upd} = D + 1$	*1 028*
Phase de compétition	$T_{PC} = T_{cd} + T_p$	*2 120*
Phase d'adaptation	$T_{PA} = T_{bc} + T_{upd}$	*1 404*
Itération d'apprentissage	$T_{IA} = T_{PC} + T_{PA}$	*3 524*
MCPS	$\frac{L \times K \times D}{T_{PC}} \times f \cdot 10^{-6}$	*30 913*
MCUPS	$\frac{L \times K \times D}{T_{IA}} \times f \cdot 10^{-6}$	*18 597*

D'autre part, les performances de la carte auto-organisatrice sont souvent exprimées par deux métriques : *Million Connection Per Second* (*MCPS*) et *Million Connection Updates Per Second* (*MCUPS*). Dans le tableau 3.8, les performances de l'architecture proposée réalisant une structure de 16×16 *neurones* utilisant des vecteurs de *dimension 256* sont présentés. On remarque que les performances de l'architecture proposée dépendent des paramètres de la carte utilisée. En effet, les courbes des figures 3.20 et 3.21 présentent l'évolution des performances de l'architecture proposée en fonction de la taille de la couche compétitive (figure 3.20) et en fonction de la dimension du vecteur d'entrée (voir figure 3.21).

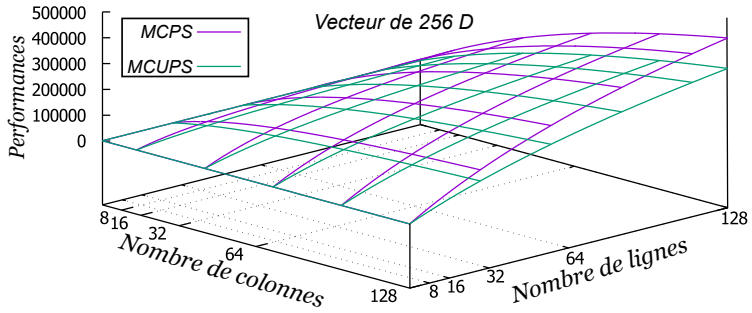


FIGURE *3.20 – Performances de l'architecture proposée en fonction de la taille de la couche compétitive*

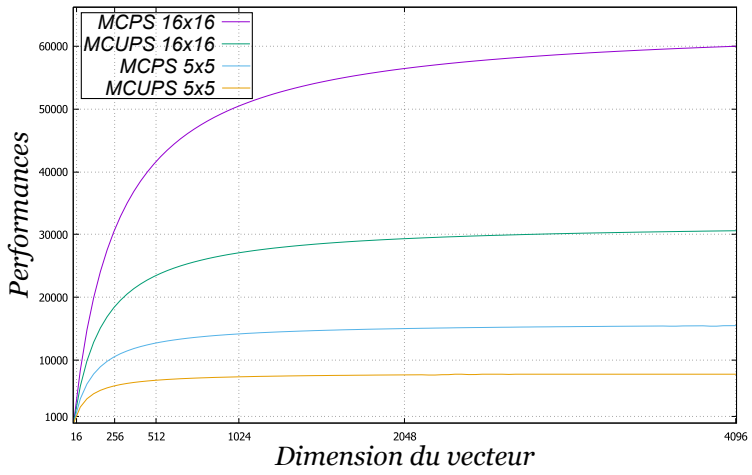


FIGURE *3.21 – Performances de l'architecture proposée en fonction de la dimension du vecteur d'entrée*

En observant la figure 3.20, on remarque que les performances augmentent progressivement avec la taille de la couche compétitive. Dans l'architecture proposée, la distribution du traitement global sur des neurones interconnectés permet de stabiliser la fréquence d'horloge maximale n'affectant pas ainsi le temps d'exécution total avec le nombre de neurones plus élevé utilisés en phase de compétition. De plus, l'augmentation de la dimension du vecteur d'entrée permet également d'augmenter les performances (voir la figure 3.21). D'un autre côté, comme nous pouvons le constater de la figure 3.21, à partir d'un certain niveau l'augmentation de la dimension ne permet plus d'améliorer les performances. Cela est dû au *traitement sériel* des éléments de vecteur d'entrée. Car si le temps de calcul des éléments de vecteurs (calcul de la distance et mise à jour des poids) dépasse largement le temps des opérations de communication (*propagation systolique* et *rétro propagation d'identifiant du neurone gagnant*), l'exécution en parallèle de l'algorithme de Kohonen sur les neurones distribués ne permettra plus de l'accélérer de manière significative. En effet, le traitement séquentiel des vecteurs d'entrée de très grande dimension devient prépondérant dans l'exécution de l'algorithme et influe grandement sur les performances globales de l'architecture. En conclusion, l'architecture proposée est plus adaptée à des cartes auto-organisatrices de grande taille utilisant des vecteurs d'entrée de grande dimension.

3.6 Comparaison de l'architecture SOM-NoC avec les architectures matérielles classiques de la carte SOM

L'architecture proposée a été comparée avec d'autres approches d'implémentation matérielle de la carte auto-organisatrice proposées en littérature [1, 2, 106, 111, 115, 119, 120]. Le tableau 3.9 résume cette comparaison. L'objectif de cette comparaison est de montrer que l'architecture proposée, qui a l'avantage d'être extensible et adaptable, est aussi compétitive, de point de vue performance, avec des architectures matérielles de l'état de l'art. Pour chaque implémentation, *les paramètres de la carte*, *la technologie utilisée*, *la fréquence d'horloge maximale*, les performances en fonction de *MCUPS* et la possibilité de *flexibilité* sont présentés.

TABLEAU *3.9 – Comparaison de l'architecture proposée avec les implémentations matérielles de la littérature*

Approche	Taille *SOM*	Dimension du vecteur	Architecture	Technologie	Fréquence max	MCUPS	Adaptabilité
[115]	16×16	3	Parallèle	FPGA	33	25 344	Non
[120]	16×16	128	Sérielle	FPGA	33	17 500	Non
[111]	16×16	16	Sérielle	CMOS	100	9 102	Non
[1]	100	2 048	Systolique	FPGA	148	3 467	Non
[2]	384	1 024	SIMD	RAPTOR2000	105	4 338	Off line
[119]	16×16	2 048	Sérielle	FPGA	69	6 305	Off line
[106]	6 050	194	Séquentielle	Core I7	*NA*	1 628	Dynamique
SOM-NoC	16×16	256	Systolique	FPGA	250	18 597	Dynamique

Les conclusions tirées par cette comparaison sont les suivantes :

1- La fréquence d'horloge maximale : On remarque que pour l'architecture proposée, la fréquence d'horloge est la plus élevée parmi les architectures de l'état de l'art, d'une part à cause de la technologie plus récente utilisée et d'autre part à cause de la distribution du traitement sur les neurones opérant en parallèle et étant connectés entre eux via les routeurs d'un réseau sur puce. Par conséquent, cette architecture régulière et homogène est indépendante des paramètres et de la structure de la carte *SOM*, puisque l'ajout de nouveaux neurones/routeurs n'augmente pas le chemin crique, le délai étant le responsable principal de la fréquence de fonctionnement maximale. De plus, l'architecture interne d'un neurone étant sérielle et en pipeline contribue également à cette fréquence de fonctionnement maximale élevée.

2- Les performances temporelles : L'architecture proposée est également bien placée, du point de vue performance, par rapport à l'état de l'art. L'architecture proposée étant de type MIMD offre un degré de parallélisme important qui est contrecarré par la propagation systolique d'une part, et la fréquence de fonctionnement maximale d'autre part.

3- La flexibilité : L'approche proposée permet de rendre l'architecture matérielle de la carte auto-organisatrice de Kohonen flexible et extensible. A base de cette approche, les para-

mètres et la structure de la carte utilisés peuvent être modifiés dynamiquement au cours de l'exécution, sans avoir besoin d'une nouvelle conception.

La comparaison présentée précédemment dans le tableau 3.9 permet de donner une idée sur le positionnement de notre approche par rapport à l'état de l'art. D'un autre côté, on remarque que les valeurs hétérogènes des paramètres des cartes *SOM* et technologies utilisées ne permettent pas d'effectuer une comparaison dans les mêmes conditions. C'est pour cela que nous avons développé à des fins de comparaison deux architectures matérielles classiques de la carte auto-organisatrice : les architectures *massivement parallèle* et *séquentielle* proposées respectivement dans les articles [115] de *Hikawa et Maeda* et [116] de *Hikawa et Kaida*.

Dans la première *architecture massivement parallèle*, le calcul des éléments du vecteur est effectuée en parallèle sur plusieurs étages d'opérateurs arithmétiques asynchrones au niveau de chaque neurone. Tous les traitements sont effectués en un seul cycle d'horloge de la carte *SOM* y compris la recherche du neurone gagnant s'effectuant à base d'un comparateur global. Il est à noter qu'avec cette approche, la taille maximale synthétisable testée est de 8×8 *neurones*. Donc, les résultats affichés pour les structures supérieures à 8×8 sont estimés.

La deuxième *architecture séquentielle* consiste en un seul neurone physiquement implanté. Ce module représente séquentiellement tous les neurones de la carte auto-organisatrice. De plus, les éléments des vecteurs sont analysés d'une façon sérielle sur un circuit asynchrone d'opérateurs arithmétiques.

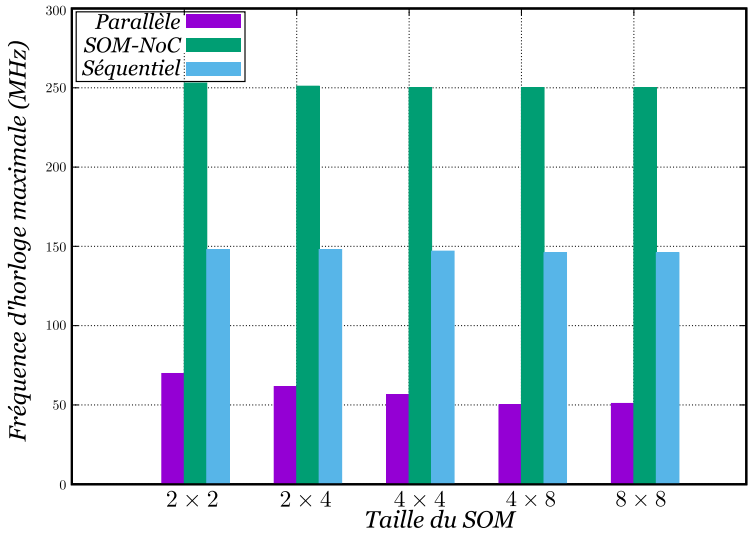


FIGURE *3.22 – Comparaison de la fréquence maximale de l'architecture proposée par rapport aux architectures classiques*

Les tests ont été effectués sur des architectures traitant des vecteurs de *dimension de 16*, afin de comparer la fréquence d'horloge maximale ainsi que le temps d'exécution d'une itération d'apprentissage. La figure 3.22 montre que la fréquence de l'architecture proposée est stable pour les différentes tailles de la carte *SOM*. Ce résultat n'implique pas que l'architecture proposée donne les meilleurs résultats en termes de performances (*MCPS* et *MCUPS*) parmi les trois architectures

testées. Il confirme seulement que l'évolution de la taille de la carte *SOM* n'a aucune influence sur la fréquence de fonctionnement de l'architecture, ceci étant essentiellement dû à la distribution du traitement sur des neurones interconnectés. La fréquence de fonctionnement maximale de l'architecture séquentielle est également stable car sa structure matérielle change légèrement en changeant la taille de la carte *SOM*. Par contre, l'architecture massivement parallèle est la plus influencée par le changement de la taille de la carte *SOM*. Sa fréquence de fonctionnement maximale diminue progressivement avec la taille de la carte *SOM*. Cela est dû à l'utilisation d'un comparateur global dont le nombre d'étages augmente avec le nombre de neurones compétitifs.

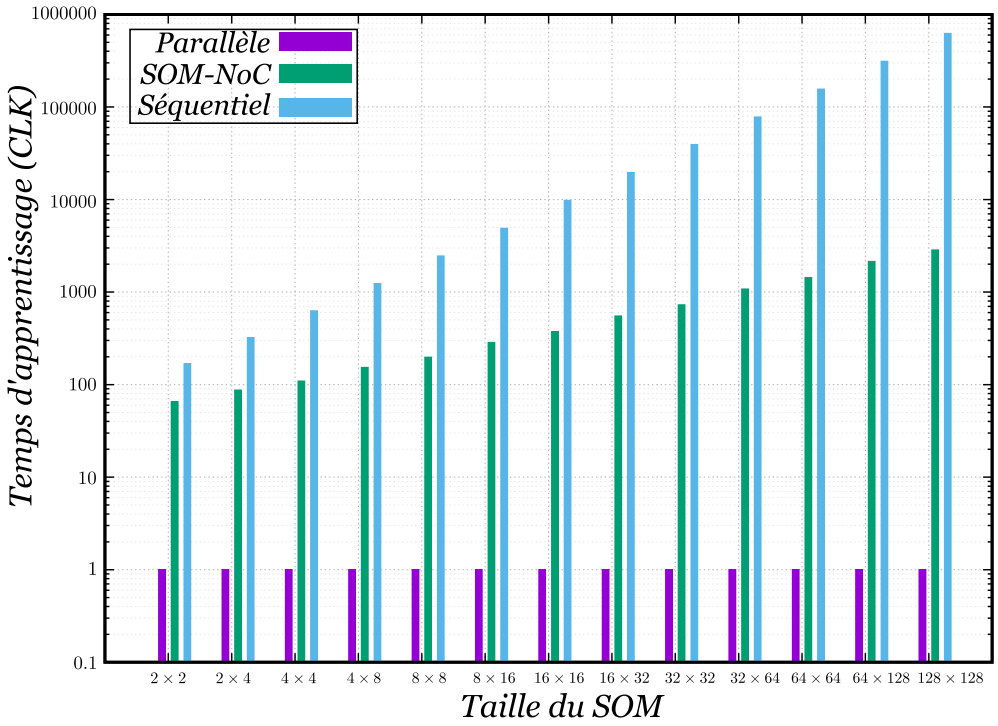


FIGURE *3.23 – Comparaison du temps d'exécution de l'architecture proposée par rapport aux architectures classiques*

La figure 3.23 montre les résultats de comparaison en termes du nombre de cycles d'horloge nécessaires, sur une échelle logarithmique, pour une itération d'apprentissage pour différentes tailles de la carte *SOM*. Ces résultats sont la véritable mesure des performances des trois architectures matérielles. L'architecture massivement parallèle est imbattable en termes de performances. Quelle que soit les paramètres de la carte *SOM* utilisés, une itération d'apprentissage est terminée en *un cycle d'horloge*. D'autre part, l'architecture séquentielle a les résultats les plus faibles, ce qui est attendu étant donné que tous les traitements sont effectués séquentiellement. L'architecture proposée de *SOM-NoC* est beaucoup plus rapide que la séquentielle, mais plus lente que la parallèle. Il convient également de préciser que l'objectif principal de notre architecture proposée est de montrer comment l'opération *SOM* peut être rendue extensible et adaptable, et non de proposer l'architecture la plus performante de la carte auto-organisatrice.

3.7 Validation de l'architecture proposée sur une application de compression d'image

Pour valider l'architecture proposée sur une application réelle, nous avons conçu un système de compression d'images à base de l'architecture proposée dans les sections précédentes. Le schéma synoptique du système de compression d'images est présenté à la figure 3.24. En littérature, on trouve plusieurs implémentations de la carte auto-organisatrice dédiées pour des applications de compression et quantification d'images [111, 115, 128, 166].

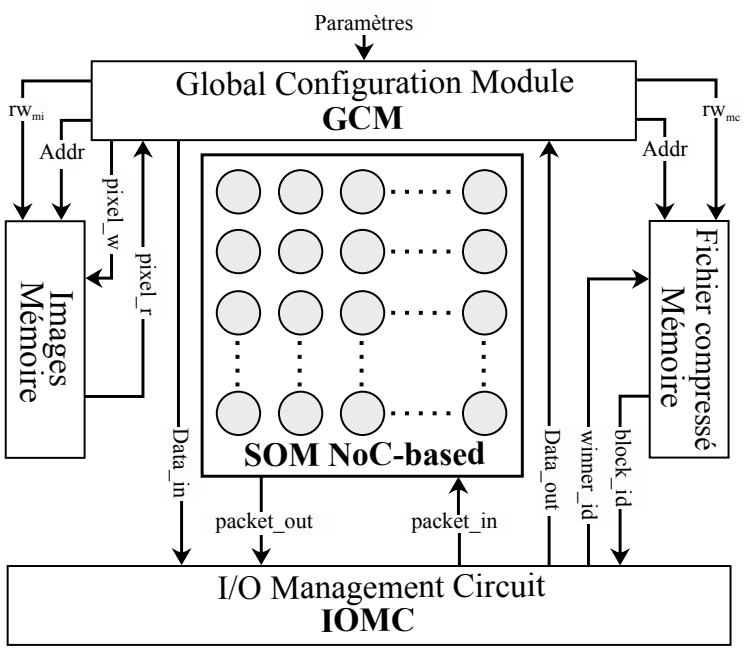


FIGURE *3.24 – Schéma synoptique du système de compression d'images*

La compression d'images en utilisant les cartes auto-organisatrices passe par deux phases : la préparation de la palette de couleurs qui représente les couleurs existantes dans l'image originale et l'indexation des pixels ou des blocs de pixels à base de la palette construite. La première phase se déroule au cours de l'apprentissage de la carte auto-organisatrice. Au cours de cette phase, une carte composée de $L \times K$ *neurones* sera alimentée par des échantillons de l'image. Un échantillon peut être composé d'*un bloc de pixels* contenant un ou plusieurs pixels voisins. Avec une image *RGB*, un pixel est présenté par trois valeurs : *rouge, vert* et *bleu*. Ces valeurs représentent les éléments du vecteur d'entrée. En effet, la dimension du vecteur d'entrée dépend du nombre de pixel $M \times M$ dans un bloc ($D = M \times M \times 3$). À titre d'exemple, un bloc de 2×2 de pixels s'il est utilisé comme un vecteur d'entrée de la carte *SOM* est représenté comme un vecteur de taille de 12 : 4 pixels fois les trois couleurs contenues dans chaque pixel. A la fin de l'apprentissage, une palette de couleurs sera construite à base de ces échantillons. Chaque élément de cette palette est représenté par un neurone sur la carte. La couleur d'un élément est présentée par le vecteur de poids du neurone.

Au cours de la deuxième phase, l'image sera balayée bloc par bloc. Donc pour une image de résolution $P \times Q$ le nombre de blocs traités N_{blk} est calculé à base de l'équation 3.14. Chaque bloc sera envoyé à la carte auto-organisatrice en phase de rappel. Le résultat de traitement d'un bloc est l'identité du neurone gagnant. Par conséquent, le résultat de la compression est un ensemble d'identité. La taille binaire de l'image compressée S_c est calculée suivant l'équation 3.15, avec *L* et *K* représentant respectivement le nombre de lignes et le nombre de colonnes sur la grille de neurones de la carte auto-organisatrice. Le taux de compression *CR* et l'espace gagné *SS* sont calculés respectivement suivant les équations 3.16 et 3.17, avec *R* étant le nombre de bits représentant un pixel.

$$N_{blk} = \frac{P \times Q}{M \times M} \tag{3.14}$$

$$S_c = N_{blk} \times [Log_2(L) + Log_2(K)] \tag{3.15}$$

$$CR = \frac{R \times P \times Q}{S_c} \tag{3.16}$$

$$SS = 1 - \frac{1}{CR} \tag{3.17}$$

La validation de notre architecture a été effectuée par trois applications de compression dont chacune utilise un jeu de paramètres différent de la carte auto-organisatrice : *A1* : 10×10 *neurones* avec un bloc de *1 pixel* (vecteur de dimension de 3), *A2* : 7×7 *neurones* avec un bloc de *1 pixel* (vecteur de 3 éléments) et *A3* : 10×10 *neurones* avec un bloc de 2×2 *pixels* (vecteur de 12 éléments). Les résultats de compression de quatre images, *Airplane*, *Lenna*, *Parrot* et *Pepper*, avec ces applications sont présentés en figures 3.25 et 3.26. Pour chaque image, on peut voir les résultats de compression à base des trois applications *A1*, *A2* et *A3*. Pour chaque application, la *palette* obtenue à la fin de l'apprentissage, *l'image reconstruite* et *la dissimilarité* entre l'image originale et reconstruite sont illustrées. On peut voir aussi, pour chaque application, les résultats numériques de la qualité de reconstruction en termes de *Peak Signal to Noise Ratio* (*PSNR*) qui représente la différence visuelle entre l'image originale et l'image reconstruite, et l'erreur quadratique moyenne *Mean Squared Error* (*MSE*) qui représente la moyenne des différences des pixels sur les deux images. D'autre part, l'espace mémoire gagné *SS* par chaque application est également représenté. On remarque que le gain en espace mémoire augmente en minimisant la taille de la carte auto-organisatrice utilisée. Cela est dû à la réduction du nombre de bits sur lesquels l'identité du neurone est codée. D'autre part, on gagne en espace mémoire en regroupant plusieurs pixels par bloc ce qui diminue le nombre des blocs traités par images. Cela provoque aussi la modification des paramètres de la carte auto-organisatrice en termes de la dimension des vecteurs.

Du point de vue qualité, on remarque que la qualité de compression dépend des caractéristiques de l'image. Par exemple, pour l'image Airplane où il n'y a pas une grande variété de cou-

leurs avec une distribution organisée des couleurs, la compression par bloc de 2×2 *pixels* donne une bonne qualité au niveau de l'image reconstruite. Par contre, si la distribution des couleurs sur l'image n'est pas organisée (on trouve une différence de couleurs entre les pixels voisins), il est préférable d'utiliser des blocs composés d'un seul pixel. D'autre part, si la variété de couleurs sur l'image est grande, il est préférable d'augmenter le nombre de neurones représentant la palette de quantification pour ne pas perdre en qualité (le cas de l'image Parrot). On peut conclure de ces résultats expérimentaux que les paramètres de la carte auto-organisatrice ne dépendent pas seule-

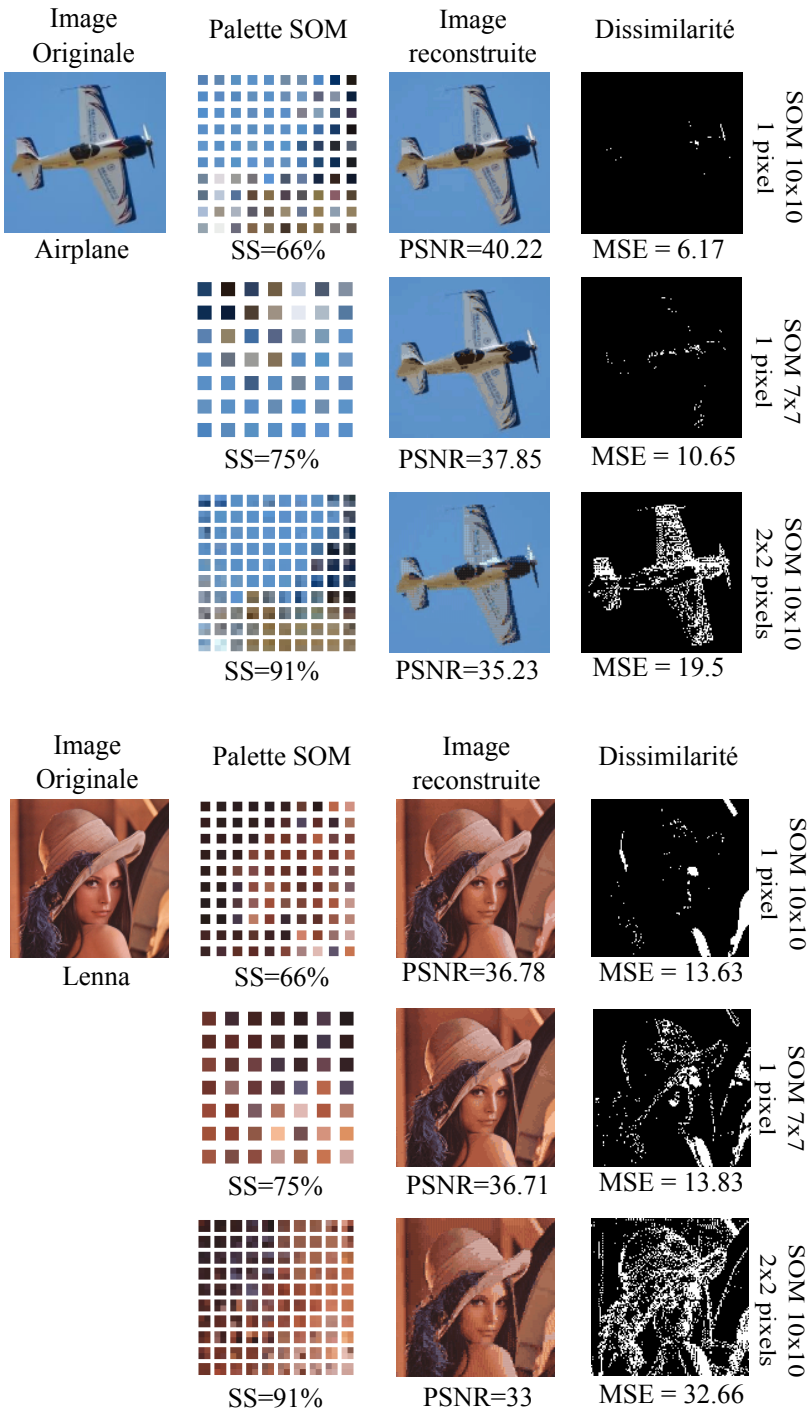


FIGURE *3.25 – Résultats de compression des images Airplane et Lenna à base de l'architecture SOM-NoC*

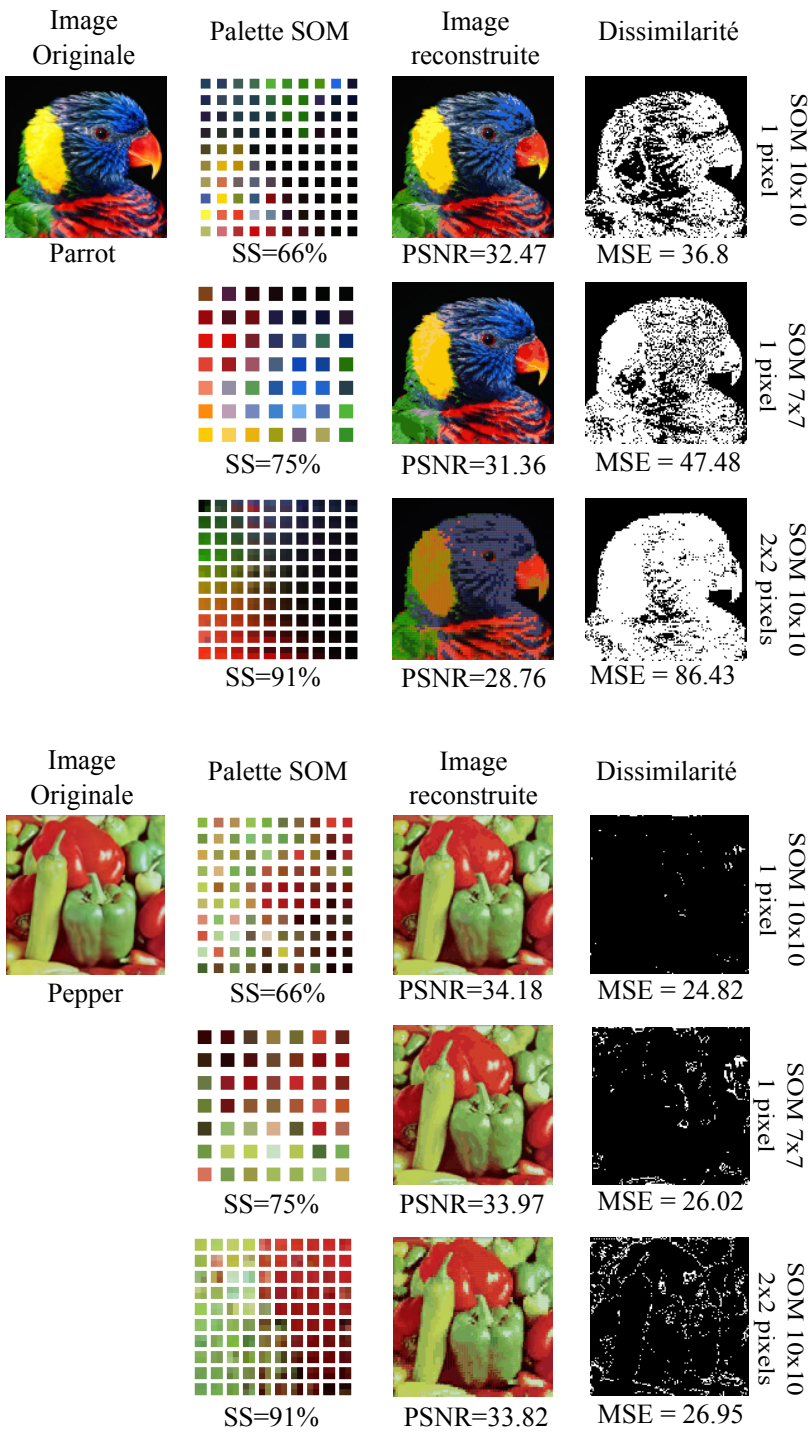


FIGURE 3.26 – *Résultats de compression des images Parrot et Pepper à base de l'architecture SOM-NoC*

ment de l'application dédiée mais également des caractéristiques de l'environnement d'entrée dans lequel le système évolue.

3.8 Limitations de l'architecture SOM-NoC proposée

L'architecture *SOM-NoC* proposée nous a permis d'introduire les propriétés de flexibilité et d'extensibilité aux implémentations matérielles de la carte auto-organisatrice. Les nouvelles propriétés ont été notamment offertes par la distribution du traitement global et l'utilisation de la technologie de communication à base des réseaux sur puce. Avec cette nouvelle approche, les opérations de l'algorithme de Kohonen peuvent s'effectuer sur différents modules indépendants et communicants de l'architecture dynamiquement reconfigurable. La synchronisation de l'exécution de l'algorithme est assurée par l'échange des messages entre les différents modules. Par conséquent, le trafic des messages dans le réseau *NoC* est important dans cette approche. Le tableau 3.10 résume la quantité de messages utilisés en fonction des paramètres de la carte *SOM*.

TABLEAU *3.10 – Quantité de messages utilisés pour une itération d'apprentissage*

Opération	Nombre de message	Nombre de flit
Distribution des éléments de vecteur d'entrée	$N_{m,x} = L \times K$	$N_{f,x} = D \times N_{m,x}$
Propagation systolique	$N_{m,p} = (L-1) \times (K-1) \times 2 + L + K - 2$	$N_{f,p} = 2 \times N_{m,p}$
Rétro-propagation d'identifiant du neurone gagnant	$N_{m,rp} = (L-1) + (K-1) \times L$	$N_{f,rp} = N_{m,rp}$

Le remplacement des communications point à point par des communications à base du *NoC* engendre une latence de communication importante. En effet, l'échange des messages qui est assuré en quelques cycles d'horloge pas des liens directs, est remplacé par des opérations de communication exécutées indépendamment sur le *NoC*. On distingue trois opérations de communication pour l'exécution d'une itération d'apprentissage dans l'architecture proposée : la distribution des éléments de vecteur, la propagation systolique pour la recherche du neurone gagnant et la rétro-propagation de l'identité du neurone gagnant. La durée d'exécution de ces opérations dépend essentiellement de la taille de la couche compétitive.

En supposant qu'une itération d'apprentissage commence à la réception du vecteur d'entrée par les neurones, le temps de distribution des éléments de vecteur d'entrée n'est pas pris en compte pour le calcul des performances. Dans ce cas, la distribution du temps d'exécution d'une itération d'apprentissage sur les différentes opérations dépendra seulement de la taille de la couche compétitive. La figure 3.27 présente la distribution temporelle des opérations en fonction de la taille de la carte *SOM*. On remarque que seulement le temps d'exécution des opérations de communication augmente avec la taille de la carte *SOM*. A un certain niveau, le temps d'exécution des opérations de traitement (calcul de la distance et adaptation des poids) devient négligeable. Il faut noter que pendant ce temps de communication les modules de traitement restent inactifs. Par conséquent, le principal inconvénient de l'architecture *SOM-NoC* réside dans le temps de communication élevée principalement causé par l'utilisation des *NoC* classiques.

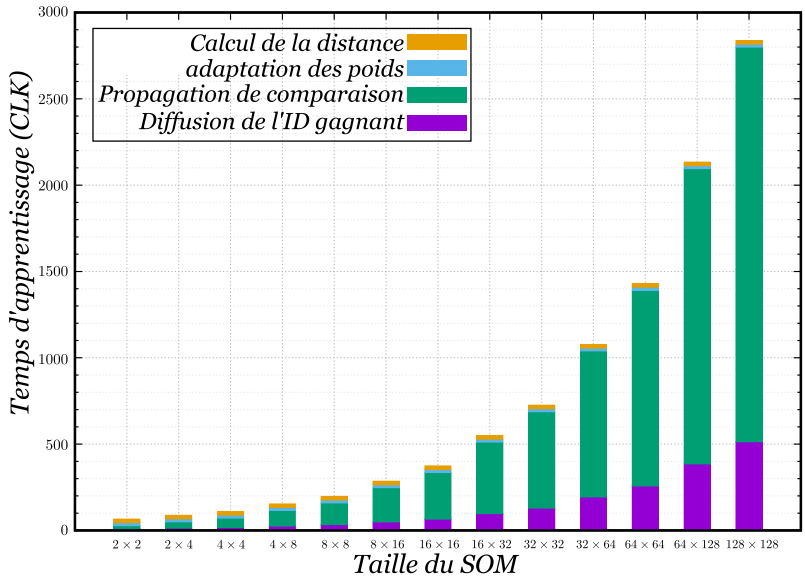


FIGURE *3.27 – Distribution du temps d'exécution des opérations dans l'architecture proposée*

3.9 Conclusion

Dans ce chapitre, nous avons présenté une architecture matérielle flexible et extensible de la carte auto-organisatrice de Kohonen tout en respectant la propriété de performance des implémentations matérielles. Dans cette architecture, nous avons choisi de distribuer le traitement de l'algorithme de Kohonen en éliminant les modules de traitement global sanctionnant l'extensibilité des architectures matérielles classiques de la carte auto-organisatrice. D'autre part, nous avons séparé les opérations de traitement de l'algorithme de Kohonen de celles de communication afin d'éliminer la dépendance architecturale entre ces opérations. Par conséquent, l'architecture proposée a été présentée sur deux couches : une couche de traitement et une couche de communication. Dans la couche de communication, la technique de communication classique point à point souvent utilisée a été remplacée par la technique de communication à base des réseaux sur puce. Cela a permis d'offrir plus de flexibilité et d'extensibilité à l'architecture proposée.

L'architecture proposée a été validée sur l'exemple d'un système de traitement d'image appliqué à la compression d'images. Les résultats obtenus ont montré que les paramètres de la carte ne dépendent pas uniquement de l'application, mais également des caractéristiques de l'environnement d'entrée. Cette constatation met au premier plan une question souvent posée par les concepteurs de cartes *SOM* : comment peut-on fixer à l'avance les paramètres d'une carte *SOM* dédiée à une application spécifique ? Ou encore, comment peut-on adapter dynamiquement une carte *SOM* et ses paramètres si on ne connait pas l'application et ses propriétés à l'avance ? Dans le chapitre suivant, nous allons essayer de répondre à ces questions en proposant deux nouvelles architectures matérielles de la carte *SOM* basées sur l'architecture *SOM-NoC* présentée dans ce chapitre.

Chapitre 4

Architecture matérielle auto-adaptative de la carte auto-organisatrice

Sommaire

"*Le plus grand génie est celui qui accueille tout en lui, qui sait s'adapter à tout sans faire le moindre tort au fond particulier que l'on nomme caractère, au contraire en l'exaltant et en l'améliorant.*"

Wolfgang Von Goethe

4.1 Introduction

Dans le chapitre précédent, nous avons présenté une architecture matérielle de la carte *SOM* permettant de résoudre les principales limitations des architectures matérielles de l'état de l'art. La nouvelle architecture matérielle de la carte *SOM* proposée dans ces travaux de recherche offrent des propriétés d'extensibilité et de flexibilité à une carte *SOM* matérielle, non seulement au niveau global où le nombre de neurones de la carte et la dimension de la couche d'entrée peuvent être paramétrés de manière dynamique, mais également au niveau local au niveau de chaque neurone, où la dimension de son vecteur de poids et ses neurones voisins avec lesquels l'échange de données pour déterminer le neurone gagnant local s'effectue, peuvent également être définis à la volée. Ces nouvelles propriétés ont été rajoutées à l'architecture matérielle de la carte *SOM* grâce à la séparation de la couche de calcul, représentée par les neurones, et la couche de communication assurant l'échange de données entre les neurones eux-mêmes et entre la couche d'entrée et tous les neurones de la carte. Dans l'architecture proposée dans ces travaux de recherche, la couche de communication est réalisée à l'aide d'un réseau sur puce *NoC*, dont une des caractéristiques principales est la propriété d'extensibilité. Dans le chapitre précédent, nous avons démontré comment dans l'architecture proposée tous les paramètres d'une carte *SOM* peuvent être changés de manière dynamique, même au cours du fonctionnement de l'architecture. L'architecture a également été comparée avec les architectures matérielles les plus récentes de l'état de l'art. Cette comparaison a permis de démontrer que l'ajout de ces nouvelles propriétés n'affectent pas de manière significative les performances d'exécution des opérations de l'algorithme de *SOM* dans l'architecture proposée et qu'elle trouve bien sa place dans le paysage des architectures matérielles performantes de la carte *SOM*.

D'un autre côté, l'architecture proposée basée sur un réseau sur puce *NoC* utilise la manière systolique pour propager de neurone en neurone la distance minimale et l'identité du neurone gagnant. Cette approche d'échange systolique dépend également de la taille de la carte *SOM* à réaliser à un moment donné : plus la taille de la carte *SOM* est élevée, plus longue est l'étape de recherche du neurone gagnant et par conséquent plus inactifs sont les neurones de la carte *SOM* qui attendent de connaitre l'identité du neurone gagnant pour terminer l'itération d'apprentissage en cours et passer à la suivante. Dans ce chapitre, nous proposons une solution pour rendre les neurones de l'approche *SOM-NoC* plus actifs et occupés d'une part et pour réaliser en même temps plusieurs structures *SOM* caractérisées par différents paramètres d'autre part. De surcroit, nous présentons également dans ce chapitre une solution permettant de réaliser des cartes *SOM* de manière dynamique. A notre connaissance, c'est la première architecture matérielle de la carte *SOM* dynamique jamais présentée dans la littérature.

4.2 Système multi-application à base de l'architecture SOM-NoC

Afin de résoudre les problèmes de l'architecture *SOM-NoC* évoquée précédemment, nous proposons dans cette section une approche permettant de réduire le temps d'inactivité des neurones de la carte *SOM*. L'approche proposée consiste en un *système multi-applications* à base de l'architecture *SOM-NoC*. Étant donné la propriété de l'architecture matérielle *SOM-NoC* lui permettant de s'adapter dynamiquement à différentes applications à base des *SOM* utilisant des jeux de paramètres différents, l'idée de base est d'utiliser la même architecture matérielle pour exécuter simultanément plusieurs cartes auto-organisatrices dont les paramètres sont indépendants. Dans la nouvelle architecture présentée dans cette section, le temps d'inactivité des neurones de l'architecture sera exploité pour exécuter plusieurs algorithmes de Kohonen en pipeline. La nouvelle architecture permettant de commuter entre différentes cartes auto-organisatrices, est appelée *architecture SWitch APplication Self-Organizing Map (SWAP-SOM)*.

Afin de pouvoir assurer l'exécution de l'algorithme de Kohonen pour différentes cartes ayant des jeux de paramètres différents, les paramètres d'exécution de chaque carte doivent être sauvegardés : le nombre de neurones utilisés dans une carte *SOM* ainsi que les vecteurs de poids de chaque neurone appartenant à cette carte auto-organisatrice. C'est au cours de l'exécution de l'architecture que la carte *SOM* à utiliser à un moment donné doit être identifiée de manière dynamique. Au niveau de chaque neurone, la reconfiguration et commutation d'un jeu de paramètres à un autre doit se faire de manière rapide sans pénaliser les performances globales de l'architecture.

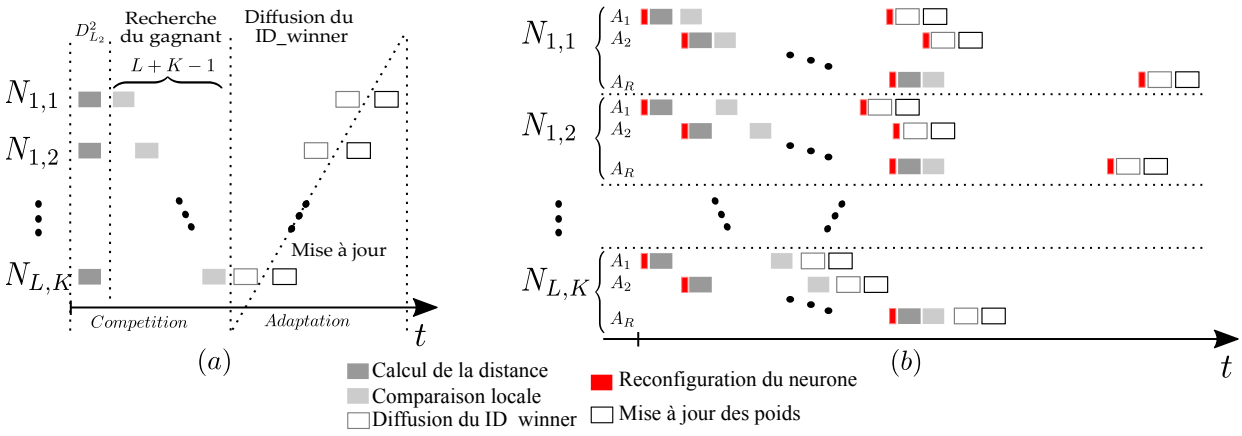


FIGURE *4.1 – (a) Chrono-grammes d'exécution d'une carte SOM sur l'architecture SOM-NoC ; (b) Chronogrammes d'exécution en pipeline de plusieurs cartes SOM sur l'architecture SWAP-SOM*

La figure 4.1 illustre une comparaison entre les chrono-grammes d'exécution de l'algorithme de Kohonen dans l'architecture de base *SOM-NoC* et ceux *d'exécution en pipeline* de différentes applications dans l'architecture *SWAP-SOM*. On remarque que dans l'architecture *SOM-NoC* (la figure 4.1.a), les opérations de l'algorithme de Kohonen sont exécutées *de manière séquentielle* dans un ordre chronologique. Après le calcul de la distance pour un vecteur d'entrée, le neurone

doit attendre la fin de la *propagation systolique* pour connaitre l'identité du neurone gagnant, qui sera ensuite diffusée vers tous les neurones avant la mise à jour de leurs poids. Le prochain vecteur d'entrée ne pourra être traité qu'après l'adaptation des poids de tous les neurones par rapport au vecteur d'entrée en cours. En effet, l'attente au sein d'un neurone n'est pas causée par son occupation à cause des calculs intensifs, mais plutôt par la dépendance des données au sein de l'architecture, et montre clairement une utilisation inefficace des neurones au sein de l'approche *SOM-NoC*.

Les chrono-grammes d'exécution de plusieurs applications à base de *SOM* dans l'architecture *SWAP-SOM* sont présentés en figure 4.1(b). On remarque que dans l'architecture *SWAP-SOM*, un neurone est beaucoup plus occupé que dans l'architecture *SOM-NoC*. En effet, un neurone de l'architecture *SWAP-SOM* enchaine les vecteurs d'entrée appartenant à différentes applications $A_i (i \in [1\,, R]$, où R est le nombre d'applications), caractérisées par des paramètres différents (nombre de neurones, dimension des vecteurs d'entrée/de poids). Cet enchainement s'effectue après un temps de reconfiguration permettant à un neurone d'adapter ses propres paramètres à l'application en cours et d'effectuer les quatre opérations de l'algorithme de Kohonen en pipeline. Pour assurer cette exécution en pipeline, l'architecture doit être capable de commuter entre les différentes applications de manière rapide. Dans la suite de cette section, nous présenterons les choix architecturaux effectués permettant d'adapter l'architecture de base *SOM-NoC* à un traitement multi-applications.

4.2.1 Architecture SWAP-SOM

La figure 4.2 présente le schéma synoptique de l'architecture *SWAP-SOM*. Dans cette architecture, le module *Global Configuration Management* (*GCM*) prend comme entrée l'identifiant de l'application sur l'entrée *Application*, tandis que les paramètres de chaque application sont envoyés sur l'entrée *Paramètres* avec un signal de validation sur l'entrée *New_appl*. Les paramètres de chaque application seront enregistrés dans deux mémoires adressées par l'identifiant de l'application au sein du module *GCM*. La première est réservée pour stocker les dimensions du vecteur d'entrée. Ces informations serviront à assurer le bon nombre d'éléments d'entrée qui doivent être envoyés sur la couche d'entrée de l'application en cours. La deuxième mémoire est réservée pour l'enregistrement de la taille de la couche compétitive de chaque application afin d'identifier le nombre et les adresses des neurones concernés par la réception des données d'entrée et d'échange entre neurones.

4.2.2 Architecture modifiée du neurone

L'architecture d'un neurone dans l'architecture *SWAP-SOM* a été légèrement modifiée par rapport à celle du neurone de l'architecture *SOM-NoC*. Les modifications ont notamment été apportées au niveau des deux modules : *Local Configuration Management* (*LCM*) et *Vector Element Processor* (*VEP*). Au niveau du module *VEP*, les mêmes blocs mémoires ont été utilisés. Par

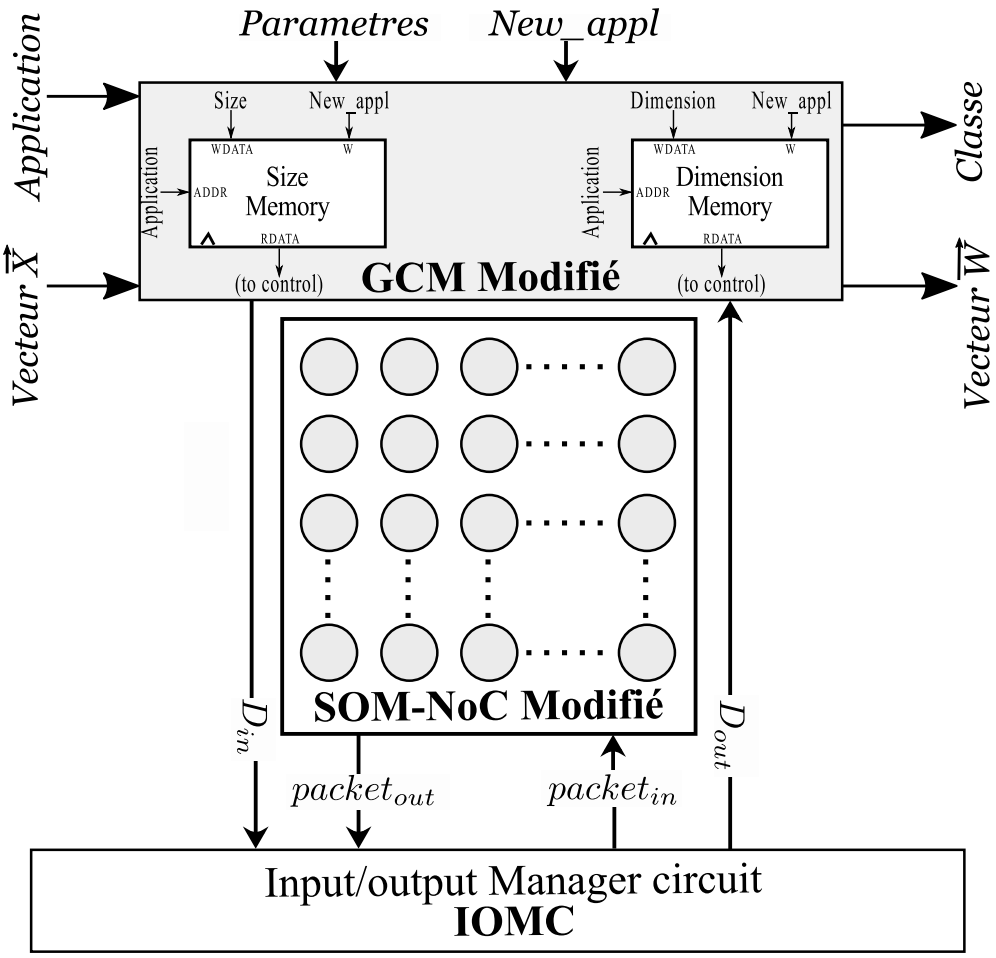


FIGURE *4.2 – Architecture SWAP-SOM*

contre, pour chaque application une zone mémoire dédiée est réservée qui est identifiée par le premier élément du vecteur d'entrée nommée *Offset*. Afin de pouvoir pointer vers la zone mémoire contenant les vecteurs de poids des neurones appartenant à l'application en cours d'exécution, l'unité de génération des adresses au niveau du *VEP* a été modifiée. L'indice de l'élément du vecteur d'entrée en cours de traitement est additionné à l'*offset* de la zone mémoire du vecteur de poids, afin de faire correspondre les éléments du vecteur de poids de l'application en question aux éléments du vecteur d'entrée. La figure 4.3 montre l'architecture interne du nouveau générateur d'adresse utilisé dans le neurone *SWAP-SOM*.

Afin d'accélérer l'opération de reconfiguration au niveau du neurone, le module *LCM* a également été modifié. La figure 4.4 présente la nouvelle architecture de ce module. Dans ce module, quatre blocs mémoires sont utilisés. Les *mémoires size* et *dimension* sont utilisées pour enregistrer *les paramètres d'applications*, tandis que la *mémoire offset* est réservée pour l'enregistrement de l'offset nécessaire pour retrouver le vecteur des poids d'une application, le paramètre étant utilisé par le générateur d'adresse du *VEP* dans les opérations de calcul de distance et de mise à jour. À la réception d'une nouvelle configuration, le contrôleur LCM calcule l'offset de la prochaine application à exécuter en se basant sur la dimension du vecteur de poids de l'application en cours

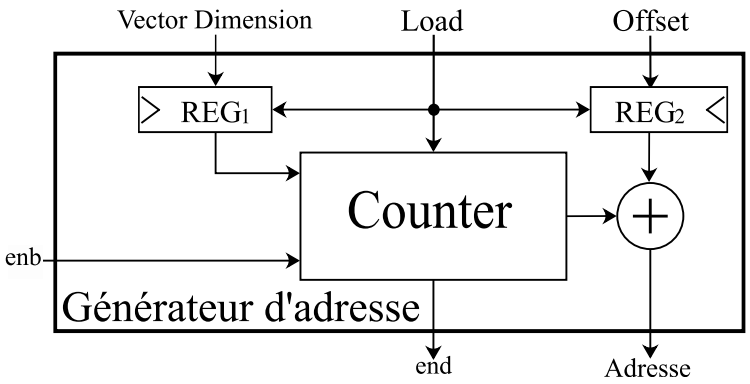


FIGURE *4.3 – Nouvelle architecture du générateur d'adresse pour le SWAP-SOM*

d'exécution. De plus, la *mémoire Update* est utilisée pour enregistrer les paramètres d'apprentissage (*taux d'apprentissage* $\beta(t)$ et *rayon de voisinage R(t)*) pour chaque application. Comme nous pouvons le remarquer d'après le signal *Upd_param* envoyé vers le module *Update Signal Generator* (*USG*), les circuits de mise à jour de ces paramètres en fonction de la progression de l'apprentissage ont été déplacés du module *USG* vers le contrôleur du module *LCM*.

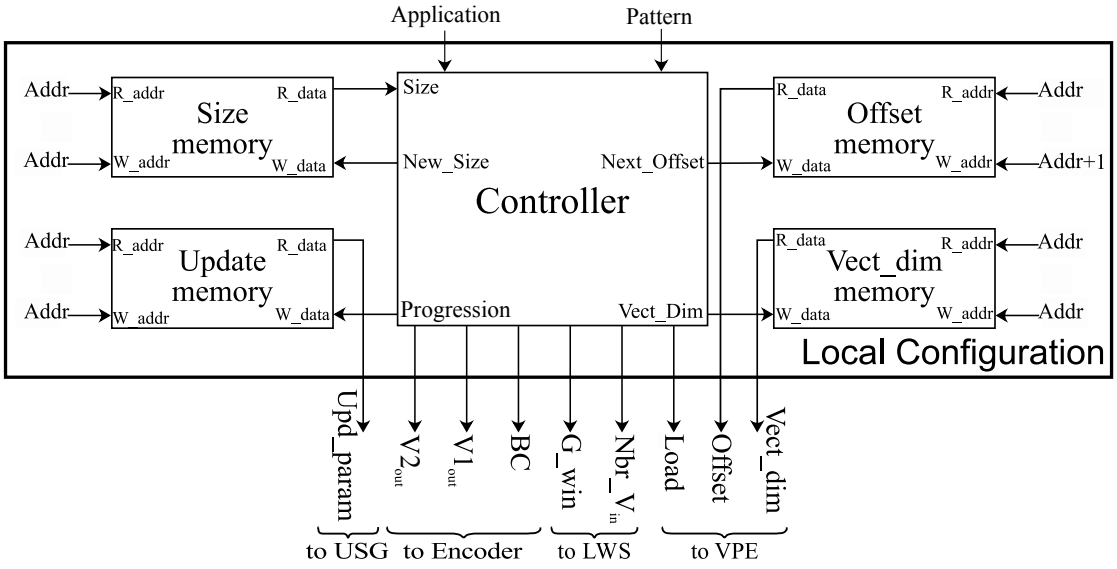


FIGURE *4.4 – Nouvelle architecture du module LCM utilisée dans l'approche SWAP-SOM*

Avec cette nouvelle architecture, pour chaque nouvelle application, les neurones associés à la carte auto-organisatrice utilisée reçoivent un message de configuration contenant le type de paramètres à utiliser à un moment donné. Après la réception de cette nouvelle configuration, le neurone prépare les paramètres de la nouvelle application et les enregistre dans ses mémoires dans les zones mémoires réservées à l'application donnée. D'autre part, chaque message circulant dans le *Network-on-Chip* (*NoC*) porte également l'identifiant de l'application associée. En effet, à la réception d'un message, le neurone analyse l'identifiant de l'application. Si cet identifiant ne correspond pas à celui de l'application actuellement traitée, le module *LCM* procède à une opération de configuration qui dure *un seul cycle d'horloge*. A la fin de cette opération, tous les

modules du neurone recevront des signaux leur permettant d'adapter leurs comportements à la nouvelle application demandée.

4.2.3 Résultats de validation de l'architecture SWAP-SOM

La nouvelle architecture a été décrite en langage *VHDL* et synthétisée pour l'implantation sur une carte *FPGA VC707 Virtex-7* en utilisant l'outil de conception *ISE 14.7 de Xilinx*. La fréquence maximale de fonctionnement obtenue atteint *250 MHz* indépendamment de la structure de la carte implantée, notamment grâce à l'extensibilité de l'approche *SOM-NoC* sur laquelle l'architecture *SWAP-SOM* est basée.

Le tableau 4.1 illustre les expressions de temps d'exécution des opérations effectuées au cours d'exécution de l'architecture *SWAP-SOM* : avec *D* étant la dimension du vecteur ; *L* et *K* étant respectivement le nombre des lignes et des colonnes sur la grille de la couche compétitive ; T_r étant le délai de passage d'un message par routeur égal à *2 cycles d'horloge* ; et T_s étant le temps de paquetage ou dépaquetage égal à *un cycle d'horloge*. Il est à noter que le temps d'exécution d'une itération d'apprentissage $\mathrm{T_{IA}}$ est égal à la somme des temps d'exécution de l'ensemble des opérations de communication et de traitement ($T_{IA} = T_{Is} + T_{cd} + T_P + T_{RP} + T_{upd}$). Par conséquent, pour une application spécifique, la *cadence* d'envoi des échantillons au cours de l'apprentissage doit être supérieure à T_{IA}. D'autre part, les opérations de diffusion de l'identité du neurone gagnant et la mise à jour des poids ne seront pas exécutées pour une itération de rappel. Donc le temps d'exécution d'une itération de rappel est $T_{IR} = T_{Is} + T_{cd} + T_P$. D'où, la *cadence* d'envoi des stimuli au cours de la phase de rappel doit être supérieure à T_{IR}. Les performances temporelles en termes de *MCPS* et *MCUPS* d'une carte auto-organisatrice composée de 16×16 *neurones* avec un vecteur de poids d'une *dimension de 256* sont respectivement *22 413 MCPS et 15 017 MCUPS*, en prenant en compte le temps nécessaire pour la distribution des éléments de vecteur d'entrée.

TABLEAU *4.1 – Expressions temporelles des opérations de l'architecture SWAP-SOM*

Type d'opération	**Opération**	**Equation** (clk)
Configuration	Configuration de la carte SOM	$T_{conf} = (T_s + T_r) \times (L + 1) + 3$
	Commutation d'application	$T_{switch} = 1$
Traitement	Calcul de la distance	$T_{cd} = T_{switch} + D + 3$
	Mise à jour des poids	$T_{upd} = T_{switch} + D + 1$
Communication	Distribution des vecteurs d'entrée	$T_{Is} = (T_s + T_r) \times (L + 1)$
	Propagation systolique	$T_P = (4 \cdot T_s + 5 \cdot T_r) \times (L + K - 2)$
	Rétro-propagation de l'identité du neurone gagnant	$T_{RP} = (L + K + 2)(T_s + T_r)$

D'autre part, l'exécution de plusieurs applications en pipeline, comme expliqué précédemment, dans l'architecture *SWAP-SOM* peut être effectuée par l'envoi successif de vecteurs d'entrée pour chaque application, avec une cadence d'envoi par application supérieure à T_{IA}. Par

conséquent, le temps total d'exécution des itérations d'apprentissage sur les différentes cartes *en pipeline* $T_{pipeline}$ est calculé suivant l'équation 4.1 :

$$T_{pipeline} = \left(\sum_{i=1}^{R-1} D(A_i) \right) + T_{IA}(A_R) \tag{4.1}$$

avec A_i étant l'identifiant de l'application. Par conséquent, les performances de deux cartes auto-organisatrices, composées chacune de 16×16 *neurones* avec un vecteur de poids de *dimension de 256*, s'élève à *33 200 MCPS et 24 327 MCUPS*.

Bien que les performances de l'architecture proposée montrent des meilleurs résultats par rapport à l'architecture *SOM-NoC* d'une part et permettant de traiter plusieurs applications en même temps d'autre part, un des objectifs était d'optimiser le taux d'utilisation/occupation des circuits de calcul au sein d'un neurone dans l'architecture de base *SOM-NoC*. La figure 4.5 présente le taux d'occupation d'un neurone au sein de l'architecture *SWAP-SOM* en fonction du nombre de cartes *SOM* (ou applications) traitées à un moment donné. Selon les résultats présentés en figure 4.5, le taux d'occupation augmente avec le nombre de cartes auto-organisatrices à traiter en même temps. On remarque également que le taux d'occupation d'un neurone augmente avec la dimension du vecteur d'entrée/poids. Cela est dû au traitement sériel des éléments de vecteur d'entrée sur chaque carte permettant d'enchainer le traitement sériel des éléments des vecteurs d'entrée/poids appartenant à des applications différentes, séparés uniquement avec un temps de reconfiguration d'un cycle d'horloge entre deux applications différentes.

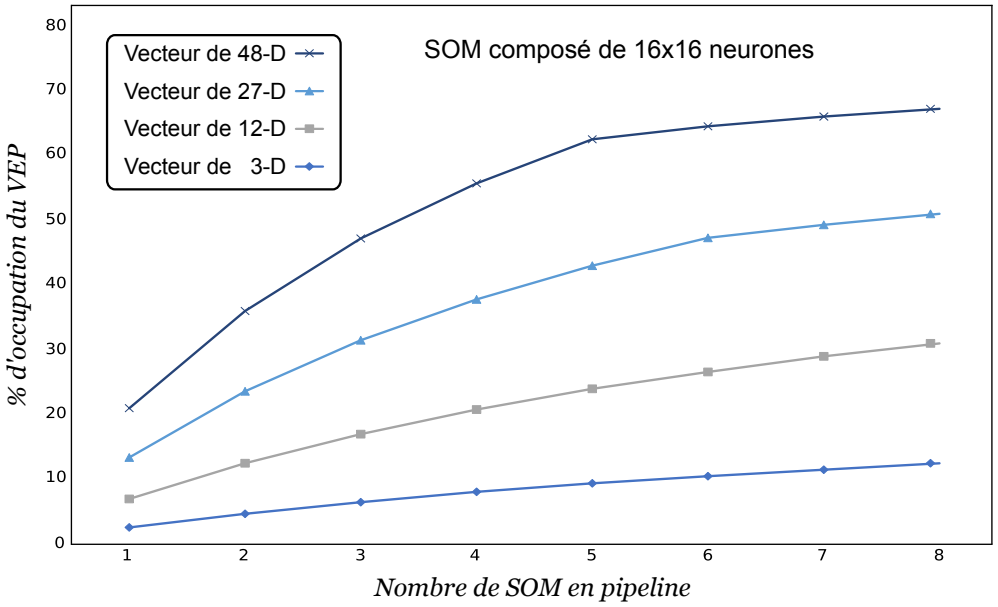


FIGURE *4.5 – Taux d'occupation de VEP en fonction du nombre de SOM exécutées en pipeline*

L'architecture *SWAP-SOM* a été validée sur un exemple de système de compression d'image avec 6 applications de compression à base des cartes auto-organisatrices ayant des paramètres différents : *A1* avec 6×6 *neurones* dont les poids représentent *1 pixel*, *A2* avec 9×9 *neurones*

dont les poids représentent *1 pixel*, *A3* avec 15×15 *neurones* dont les poids représentent *1 pixel*, *A4* avec 9×9 *neurones* dont les poids représentent *4 pixels*, *A5* avec 12×12 *neurones* dont les poids représentent *4 pixels* et *A6* avec 15×15 *neurones* dont les poids représentent *4 pixels*. L'architecture globale du système de compression est présentée en figure 4.6.

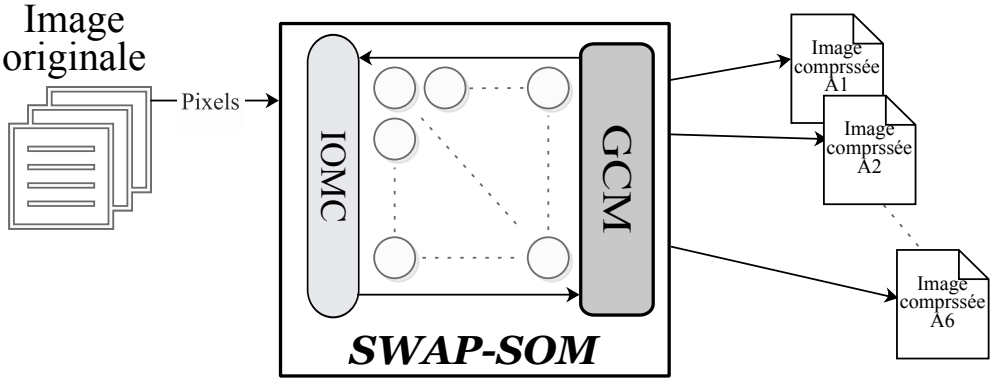


FIGURE *4.6 – Schéma synoptique du système de compression multi-application à base de l'architecture SWAP-SOM*

L'objectif principal de ce système de compression est de trouver la carte *SOM* et ses paramètres correspondants permettant d'obtenir le meilleur rapport *qualité/taux* de compression pour chaque image compressée. La qualité de la compression s'exprime avec le *Peak Signal to Noise Ratio* (*PSNR*) de l'image reconstruite. Cette métrique signifie la différence visuelle entre l'image originale et l'image reconstruite. Le *PSNR* de l'image reconstruite augmente lorsque la différence entre les couleurs de ses pixels et ceux de l'image originale est moins significative. Par contre, le taux de compression dépend non seulement du nombre de neurones utilisés pour la compression mais également du nombre de pixels représentés par un neurone. Un meilleur taux de compression est obtenu avec un minimum de neurones pouvant représenter un maximum de pixels. Les résultats obtenus par ce système de compression d'image sont présentés à la figure 4.7.

Afin de comparer l'architecture *SWAP-SOM* avec l'architecture *SOM-NoC* [167] et l'architecture *SOM-NoC adaptable* [168], nous avons développé le même système de compression d'image proposé précédemment à base de ces deux approches. Pour la première approche, 6 cartes auto-organisatrices ayant des paramètres et des structures différents ont été implantées. Chaque carte est responsable de l'exécution d'une application $A_i (i \in [1, 6])$. Par conséquent, les paramètres des cartes utilisées sont fixés au cours de la phase de conception. Au cours du fonctionnement du système, les applications sont exécutées simultanément sur des supports matériels distincts. Le système implémenté à base de l'architecture *SOM-NoC* adaptable consiste en une implantation matérielle d'une carte auto-organisatrice composée de 15×15 *neurones* avec un vecteur de dimension maximale égale à *12*. Cette implémentation peut s'adapter aux besoins de 6 applications via une reconfiguration effectuée au cours de l'exécution du système. De cette façon, les 6 applications ont été exécutées successivement sur le même support matériel avec une opération de reconfiguration effectuée après la récupération des résultats de compression d'une application A_i et avant de commencer la phase d'apprentissage de l'application A_{i+1}.

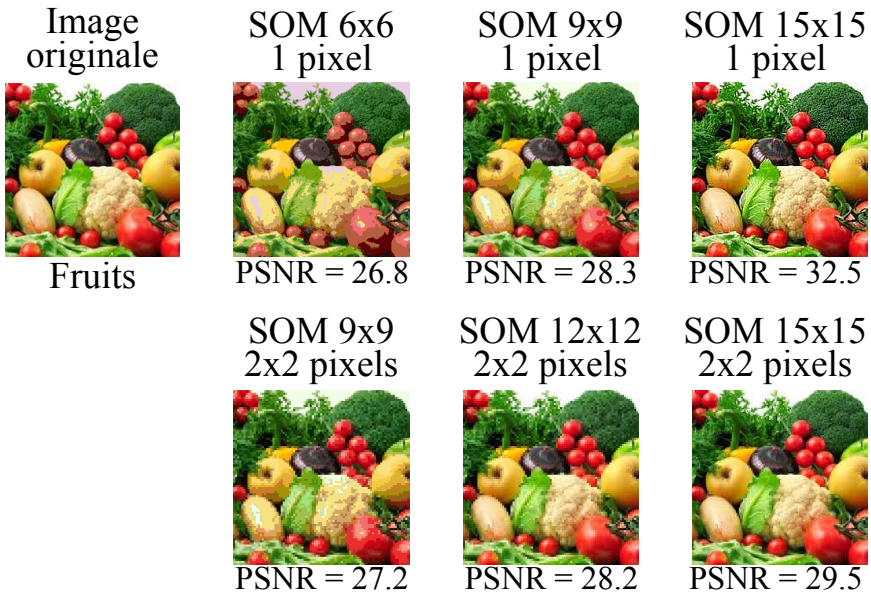


FIGURE *4.7 – Résultats de compression d'image à base de l'architecture SWAP-SOM*

Les résultats de cette expérience sont illustrés dans le tableau 4.2. Les ressources matérielles utilisées sont présentées pour chaque approche matérielle. On remarque que l'approche à base de multiples *SOM-NoC* est la plus gourmande en ressources matérielles nécessaires. Cela est dû à l'implantation de 6 différentes structures indépendantes de l'architecture *SOM-NoC* pour chaque application sans partager aucune ressource matérielle. Par contre, pour l'approche *SOM-NoC* adaptable, seulement les ressources matérielles d'une carte auto-organisatrice composée de 15×15 *neurones* avec un vecteur de *dimension de 12* sont utilisées. Ces ressources sont réutilisées par toutes les applications $\mathrm{A_i}$ ($\mathrm{i} \in [1,6]$). D'un autre côté, dans l'architecture *SWAP-SOM*, l'architecture globale et tous ses éléments sont également réutilisés par toutes les applications. On remarque l'utilisation des ressources matérielles supplémentaires par rapport à l'architecture *SOM-NoC* adaptable, notamment un nombre de registres supplémentaires parce que chaque application doit enregistrer ses vecteurs de poids pour une réutilisation ultérieure.

TABLEAU *4.2 – Comparaison de l'architecture SWAP-SOM avec Multiple SOM-NoC et SOM adaptable*

Approche	**Ressources matérielles**		**Temps d'exécution**	**Taux d'occupation**	**Réutilisation**
	# Registres	**# LUTs**			
Multiple SOM-NoC [167]	1 473 912	1 732 104	402 Clk	6.8 %	Rappel/reconstruction
SOM adaptable [168]	419 725	492 300	1 882 Clk	6.3 %	Apprentissage-rappel/ reconstruction externe
SWAP-SOM	752 432	561 653	554 Clk	20.5 %	Rappel/ reconstruction

En termes de temps d'exécution, l'approche à base de *multiples SOM-NoC* semble la plus performante. Cette constatation est expliquée par la conception spécifique d'une architecture pour chaque application. Par conséquent, aucune opération de configuration n'est effectuée dans cette approche contrairement aux deux autres architectures. Pour l'architecture *SOM-NoC* adaptable,

une opération de reconfiguration est effectuée avant de commencer l'apprentissage de chaque application avec une exécution séquentielle des applications. D'autre part, les configurations des cartes auto-organisatrices sont également effectuées au lancement de l'architecture *SWAP-SOM*. De plus, dans cette approche pour chaque vecteur d'entrée des opérations de commutation sont également effectuées. On remarque que le *taux d'occupation* qui ne dépasse pas les *7%* avec les approches à base de *multiples SOM-NoC* et *SOM-NoC adaptable*, atteint *20.5%* avec l'architecture *SWAP-SOM*.

Un dernier paramètre de comparaison est affiché dans le tableau 4.2. Il consiste en la *réutilisation des cartes auto-organisatrices* et la méthode de reconstruction des images compressées. En effet, ce paramètre correspond à la capacité de l'architecture à sauvegarder les résultats de l'apprentissage en termes de vecteurs de poids adaptés à l'application. Par conséquent, la compression et la reconstruction peuvent être effectuées sans repasser par la phase d'apprentissage. On remarque que la compression avec l'approche *SOM-NoC adaptable* doit passer à chaque fois par la phase d'apprentissage. Car, le passage d'une application à une autre engendre la perte des résultats d'apprentissage des exécutions précédentes. Par conséquent, ces résultats doivent être récupérés avant la reconfiguration et enregistrés sous forme d'une palette de couleur qui sera utilisée pour la reconstruction de l'image compressée à base d'un algorithme de couleurs indexées. Par contre, dans les architectures multiples *SOM-NoC* et *SWAP-SOM*, les résultats d'apprentissage sont réservés pour chaque application, évitant de passer par une nouvelle phase d'apprentissage comme dans le cas de l'architecture *SOM-NoC* adaptable. D'autre part, la reconstruction peut être effectuée à base de ces résultats en récupérant uniquement le vecteur de poids du neurone gagnant pour chaque vecteur d'entrée.

4.3 Les cartes SOM à structure variable au cours de l'apprentissage

Les architectures matérielles classiques de la carte auto-organisatrice sont souvent dédiées pour des applications spécifiques. Par conséquent, les paramètres de la carte sont fixés au cours de la phase de conception. Donc, comme nous avons déjà précisé, le changement de l'application impose la re-conception de l'architecture pour l'adapter aux nouveaux besoins de la nouvelle application. Cela est principalement dû au manque de flexibilité et d'extensibilité de ces architectures classiques.

Avec l'approche *SOM-NoC* que nous avons proposée dans le chapitre précédent, nous avons réussi à surmonter cette limitation en proposant une architecture flexible et extensible de la carte auto-organisatrice de Kohonen. Cette approche permet de modifier dynamiquement les paramètres de la carte pour l'adapter à n'importe quelle application. Par conséquent, il suffit d'identifier l'application ou la fonction désirée d'une application pour traduire ces besoins en termes de paramètres architecturaux et comportementaux de la carte *SOM*.

Cette opération d'identification et d'extraction des paramètres d'une application en elle-même

présente la principale limitation de la carte *SOM* pour son utilisation dans les systèmes adaptatifs à toute sorte d'applications. En effet, l'objectif est de laisser le système décider son comportement en fonction des données à traiter. D'autre part, d'après les expériences effectuées sur le système de compression d'image (voir section 3.7), on remarque que pour la même application, en changeant les caractéristiques de données en entrée (en général, les caractéristiques de l'environnement externe) la structure optimale de la carte *SOM* change. Par conséquent, on se retrouve face à la question : comment fixer la structure de la carte *SOM* au préalable avant son utilisation pour une application donnée ?

L'une des approches permettant de résoudre ce problème consiste à utiliser les réseaux de neurones évolutifs. Dans ce type de système évolutif, leur comportement est traduit par la structure du réseau de neurones utilisé qui pourra être adaptée dynamiquement aux besoins de l'application ainsi qu'aux caractéristiques de l'environnement externe, notamment au cours de la phase d'apprentissage. Dans le cas des réseaux de Kohonen, les questions principales résident dans le nombre de neurones permettant de projeter l'espace d'entrée de manière convenable tout en préservant les données les plus représentatives de cet espace d'entrée, ainsi que le nombre de neurones nécessaires pour représenter une classe de données spécifique à cet espace d'entrée. La réponse à la dernière question est partiellement proposée par la forme classique de l'algorithme *SOM* de Kohonen, où on parle de groupement de neurones ayant des propriétés similaires ou de « *Clustering* », l'opération réalisée au cours de la phase de l'apprentissage de la carte de Kohonen. Par contre, la solution à la deuxième question reste toujours d'actualité. On la repose encore une fois ici avant de présenter quelques solutions proposées dans la littérature et avant de proposer la solution matérielle présentée dans ces travaux de recherche : comment fixer le nombre de neurones capables de représenter de manière la plus optimale l'espace d'entrée d'une application donnée ?

La solution classique est d'utiliser des cartes de grandes tailles avec un nombre de neurones dépassant les besoins de l'application. Cette solution est au détriment des performances et même au détriment des résultats (le cas de taux de compression pour une application de compression d'image). Des nouvelles approches répondant aux questions posées consistent en l'utilisation de structures évolutives de la carte auto-organisatrice. Donc l'idée est d'ajouter dynamiquement des neurones au cours de l'apprentissage jusqu'à atteindre le nombre de neurones optimal permettant de projeter convenablement l'espace d'entrée. Dans ce cadre, plusieurs approches de *SOM* évolutifs ont été proposées dans la littérature, souvent sous les appellations différentes en allant de « *Growing Self-Organizing Map* (*G-SOM*) » via « *Incremental Self-Organizing Map* (*I-SOM*) » jusqu'aux structures évolutives dans le temps nommées « *Time-Varying Structure Self-Organizing Map* (*TVS-SOM*) ». La section suivante présente une synthèse de ces structures évolutives de la carte *SOM* dynamique.

4.3.1 État de l'art sur les SOM évolutifs

Les réseaux de neurones évolutifs sont des modèles neuronaux caractérisés par leur capacité de changer leurs structures à la volée ou « online », au cours de la phase d'apprentissage. Le changement de structure effectué consiste souvent en l'augmentation du nombre de neurones sur le réseau ainsi que en l'ajout ou la suppression des interconnexions entre les neurones existants. L'avantage des réseaux de neurones évolutifs réside en deux points : la résolution du problème de détermination de la taille optimale du réseau de neurones pour une application définie et l'optimisation de l'apprentissage en termes de performances temporelles et qualitatives.

Dans la littérature, nous trouvons quelques travaux proposant des approches de réseaux de neurones compétitifs et évolutifs [94, 127, 131–133, 169–176]. Ces approches reposent sur l'idée de changer la structure de la couche compétitive au cours du temps pendant l'exécution. On parle des modèles neuronaux à structure variable dans le temps ou « *Time-Varying Structure Neural Network* (*TVS-NN*) ». Étant donné leur diversité en termes d'algorithme et de manière d'évoluer, on ne trouve pas dans la littérature de description commune de ces modèles neuronaux. Dans [177], *Araujo et Rego* ont présenté une synthèse bibliographique sur les modèles neuronaux à structure variable dans le temps. Dans cet article, on y trouve une description d'un certain nombre de ces modèles en comparant leurs algorithmes ainsi qu'une présentation de domaines potentiels d'applications des modèles présentés.

L'évolution de la structure de la carte *SOM* peut être effectuée en termes de taille de la couche compétitive ou en termes de la topologie d'interconnexion entre les neurones. Dans les deux cas, la structure initiale sera modifiée au cours de la phase d'apprentissage. En 1994, *Martinetz et Schulten* [178] ont proposé un modèle neuronal évolutif en termes de la topologie d'interconnexions appelé « *Topology Representing Networks* (*TRN*) ». L'objectif de ce modèle est de projeter la topologie de l'espace d'entrée sur le modèle neuronal en adaptant les interconnexions synaptiques entre les neurones [179, 180]. De surcroit, l'algorithme d'apprentissage utilisé est différent de celui utilisé par le modèle *SOM*, étant généralement utilisé par la majorité des réseaux compétitifs. Le modèle *TRN* utilise l'algorithme « *Competitive Hebbian Learning* (*CHL*) » [181] pour adapter les connexions inter-neuronales. Néanmoins, le modèle *TRN* ne change pas le nombre initial des neurones sur la couche compétitive. Dans la même année, *Fritzke* a proposé un modèle neuronal évolutif en termes de nombre de neurones sur la couche compétitive appelé « *Growing Cell Structures* (*GCS*) » [64]. Au cours de la phase d'apprentissage, l'algorithme du modèle *GCS* change la taille de la couche compétitive en insérant de nouveaux neurones avec de nouvelles connexions sur la carte, tout en préservant la topologie de la structure initiale. D'autre part, en 1995 *Fritzke* a proposé un autre modèle neuronal évolutif en termes de structure ainsi qu'en termes de topologie [182]. A la base, le modèle proposé a été inspiré du réseau de neurones « *Neural Gas* (*GN*) » (variante inspirée et dérivée des réseaux de Kohonen) [56, 183], en lui ajoutant la propriété d'évolutivité. Ce modèle neuronal appelé « *Growing Neural Gas* (*GNG*) » permet d'ajouter de nouveaux neurones au cours de l'apprentissage avec la possibilité de modifier les connexions inter-neuronales en ajoutant ou supprimant des connexions dans la structure ini-

tiale de la couche compétitive. Depuis son introduction, le modèle *GNG* se présente comme l'un des meilleurs réseaux de neurones à apprentissage évolutif permettant de représenter rapidement la topologie de l'environnement externe ou de l'espace d'entrée.

Dans les deux modèles *GCS* et *GNG classique*, l'insertion de nouveaux neurones dans la carte est effectuée d'une manière périodique ; après un nombre prédéfini d'itération d'apprentissage, une opération d'évolution ou d'insertion sera effectuée. Bien que cette approche permette d'améliorer la qualité et le temps d'apprentissage, la structure finale en termes de nombre maximum de neurones de la carte doit être prédéterminée au préalable. De surcroit, *Marsland et al.* ont proposé un nouveau modèle neuronal évolutif appelé « *Grow When Required* (*GWR*) » permettant d'ajouter des neurones sur la couche compétitive en fonction des caractéristiques des stimuli injectés au cours de la phase d'apprentissage [184]. L'algorithme utilisé dans le modèle *GWR* insère de nouveaux nœuds chaque fois que les neurones présentés dans la structure actuelle ne permettent pas de quantifier un stimulus d'entrée avec la précision désirée. Ainsi, *GWR* permet de mieux adapter la structure de la couche compétitive à l'environnement externe que les approches *GNG* et *GCS* [177]. Cette approche a également permis un regain d'intérêt des chercheurs en *SOM* dynamiques et évolutifs en permettant de déterminer la structure optimale de la carte selon les caractéristiques de l'environnement externe d'une manière autonome [134, 185, 186]. Cela est essentiellement assuré par des critères d'arrêt permettant d'achever la phase d'apprentissage si la structure résultante répond aux besoins de l'application. Ainsi, des travaux innovants ont enrichi le modèle *GNG* initial permettant de lever les limitations d'identification de la structure appropriée, fixée initialement dans la phase de conception [186–189].

Dans la littérature, on trouve aussi des modèles neuronaux évolutifs à base des architectures hiérarchiques appelés « *Growing Herarchical Neural Network* (*GH-NN*) » [101, 190, 191]. Ces réseaux de neurones permettent de représenter les données d'entrée avec des relations hiérarchiques. Par conséquent, en cas de détection d'une relation hiérarchique de données d'entrée une nouvelle couche hiérarchique sera ajoutée à la structure initiale, tout en conservant les informations requises dans la couche supérieure. Ainsi, chaque couche du modèle *GH-NN* peut être évolutive [192, 193]. Par conséquent, en cas de similarité des données d'entrée déjà identifiées sur une couche de neurones, des neurones supplémentaires peuvent être ajoutés sur une couche hiérarchique, pour mieux représenter les données d'entrée.

Des approches évolutives de la carte auto-organisatrice ont également été proposées dans la littérature. *Qiang et al.* ont proposé dans [194], un résumé des différentes approches de *SOM* évolutif ou à structure variable au cours du temps, plus précisément les cartes *SOM* à structure croissante (*G-SOM* ou *I-SOM*). Dans ces travaux, on distingue trois types de *SOM* à structure croissante : « *Evolve Self-Organizing Map* (*E-SOM*) », « *Incremental Grid Growing Self-Organizing Map* (*IGG-SOM*) » et « *Growing Hierarchical Self-organizing Maps* (*GH-SOM*) ».

***E-SOM* :** Dans ce modèle neuronal, la structure de la couche compétitive s'incrémente en termes de nombre de neurones [170, 195]. Initialement, aucun neurone n'est présent sur la couche compétitive. La structure de la carte est générée au cours de la phase d'apprentissage avec

des neurones dont le vecteur de poids associé est de même dimension que le vecteur des stimuli présentés à l'entrée. A la réception d'un nouveau stimulus, on calcule sa similarité par rapport aux neurones existants sur la carte. En cas de dissimilarité, un neurone sera ajouté sur la couche compétitive. Ce dernier prendra en charge tous les stimuli similaires de la couche d'entrée. La topologie d'une carte *E-SOM* est généralement de type irrégulier. Par conséquent, aucune contrainte topologique n'est imposée dans cette approche évolutive. Les neurones ne sont pas organisés sur une distribution unidimensionnelle ou bidimensionnelle. La relation de voisinage entre les neurones est définie en fonction de la similarité de leurs vecteurs de poids.

***IGG-SOM* :** Contrairement à l'approche *E-SOM*, une contrainte topologique est imposée pour l'approche *IGG-SOM* [182, 196]. Les neurones d'une carte *IGG-SOM* sont distribués sur une grille bidimensionnelle. Cette topologie de distribution est respectée au cours de la phase d'apprentissage. Par conséquent, la structure de la couche compétitive sera modifiée en ajoutant un ensemble de neurones permettant d'incrémenter la dimensionnalité de la grille actuelle. Initialement, la structure de la carte est composée d'une grille de 2×2 neurones. Au cours de la phase d'apprentissage, une ligne ou une colonne sera ajoutée à la grille afin de mieux projeter l'espace d'entrée. Dans ce cas, le voisinage entre les neurones ne dépend pas seulement des similarités de leurs vecteurs de poids, mais également de leurs positions sur la grille.

***GH-SOM* :** Ce type de carte *SOM* à structure croissante est classé parmi les modèles neuronaux *GH-NN*. Il permet d'améliorer les performances des cartes *SOM* avec une architecture hiérarchique [174, 175]. Ce modèle neuronal permet ainsi de mieux représenter les données d'entrée avec des relations hiérarchiques. Initialement, la carte est composée d'une seule couche de neurones. Au cours de la phase d'apprentissage, d'autres couches peuvent être ajoutées à la structure actuelle, en cas de détection d'une relation hiérarchique entre un stimulus et une information représentée par le vecteur de poids d'un neurone. A la fin de la phase d'apprentissage, la structure finale de la carte sera composée de plusieurs couches hiérarchiquement inter-connectées.

En 2009, *Rego et Araujo* ont proposé un modèle neuronal à base de *SOM* à structure variable au cours de temps (*TVS-SOM*) [195,197]. Ce modèle appelé « *Growing Self-Reconstruction Maps* (*GSRM*) » a été développé pour créer des maillages triangulaires nécessaires pour la reconstruction 3D surfacique. Au cours de l'apprentissage dans le modèle *GSRM*, on peut distinguer deux types d'opération : l'adaptation des poids et l'incrémentation de la structure initiale. La première opération consiste en l'apprentissage classique de l'algorithme de Kohonen. Par contre, l'opération d'incrémentation permet quant à elle d'ajouter un neurone à coté du neurone le plus sélectionné au cours de la phase d'apprentissage dite classique. L'incrémentation de la structure est effectuée suivant l'approche *E-SOM* présentée précédemment. De cette manière, l'insertion d'un neurone est aléatoire en terme de position, qui ne dépend que des caractéristiques des neurones actifs dans la structure actuelle utilisée et pas de leur topologie de distribution. Par conséquent, la

topologie de la couche compétitive dans cette approche évolutive peut être irrégulière. Enfin, si la structure actuelle de la carte est composée d'un nombre maximum de neurones fixé au préalable, l'apprentissage sera achevé au moment d'atteindre cette valeur maximale de neurones.

En 2010, *Kuremoto et al.* ont proposé une carte *SOM* à structure croissante au cours de temps appelée « *ParameterLess Growing Self-Organizing Map* (*PL-G-SOM*) » [127]. Dans ce modèle neuronal, les neurones sont distribués sur une grille bidimensionnelle permettant de représenter les caractéristiques fréquentielles d'une image couleur en utilisant les ondelettes. Pour respecter la contrainte topologique de la carte, les auteurs ont utilisé l'approche *IGG-SOM*. Au cours de l'apprentissage, on procède à un apprentissage classique suivant l'algorithme de Kohonen. Si on détecte une dissimilarité signifiante, une opération d'incrémentation sera effectuée. Dans ce cas, un ensemble (ligne/colonne) sera inséré sur la couche compétitive entre le neurone gagnant et son voisin le plus éloigné en fonction de la distance entre leurs vecteurs de poids. Enfin, si la structure actuelle représente convenablement l'espace d'entrée, l'apprentissage sera achevé avec la structure actuelle de la carte. L'approche *PL-G-SOM* a été utilisée dans [94] dans un système d'interaction homme-machine destiné à la reconnaissance d'instruction des signes de la main.

En 2012, Ayadi et *al.* ont proposé dans [175] une carte *SOM* hiérarchique à structure croissante au cours de temps appelée « *Multilevel Interior Growing Self-Organizing Maps* (*MIG-SOM*) ». La structure de la carte proposée est composée de plusieurs couches compétitives représentées sur des niveaux hiérarchiques différents. Au cours de la phase d'apprentissage, de nouveaux neurones peuvent être ajoutés à la carte. Les opérations d'incrémentation sont effectuées d'une part, suivant l'approche *GH-SOM* en détectant une nouvelle relation hiérarchique entre le stimulus et le vecteur de poids d'un neurone présent sur l'une des couches hiérarchiques. Dans ce cas, une nouvelle couche sera insérée sur le niveau hiérarchique inférieur au niveau de la couche contenant le neurone gagnant. D'autre part, en cas de détection d'une dissimilarité signifiante, un neurone peut être inséré sur une couche existante suivant l'approche *E-SOM*. L'approche *MIG-SOM* a été utilisée dans une application de classification de données multi-dimensionnelles.

Les modèles neuronaux évolutifs ont permis d'obtenir de bons résultats en termes de performances temporelles et qualitatives. D'autre part, à base de ces approches évolutives les principales limitations d'une structure *SOM* où sa structure optimale et adaptée à une application spécifique doit être déterminée au préalable, ont été surmontées. Par contre, l'état de l'art sur les implémentations des réseaux de neurones évolutifs reste toujours dans le cadre des approches logicielles à cause des besoins de flexibilité et d'extensibilité pour ce type de réseaux de neurones artificiels. Dans nos travaux ultérieurs présentés dans le chapitre précédent, nous avons réussi à ajouter ces propriétés aux implémentations matérielles de la carte *SOM*. Les résultats obtenus nous ont permis de proposer une architecture matérielle innovante d'une carte *SOM* à structure variable à la volée (online). Cette architecture matérielle évolutive est présentée dans les détails dans les sections suivantes.

4.3.2 Approche Growing Grid SOM à base de l'architecture SOM-NoC

Dans cette section, nous proposons une architecture matérielle évolutive de la carte auto-organisatrice. L'évolution de l'architecture proposée consiste en l'incrémentation des dimensions de la grille de la couche compétitive de la carte en termes de lignes ou de colonnes. Cette approche a été appelée « *Growing Grid Self-Organizing Map* (*GG-SOM*) » et est présentée dans les détails dans la suite de cette section.

i) Procédure d'apprentissage de l'approche GG-SOM

Dans cette approche, la procédure d'apprentissage classique de l'algorithme de Kohonen a été modifiée. A cette procédure, des opérations intermédiaires d'incrémentation de la taille de la grille des neurones ont été ajoutées. Ces opérations consistent en ajout d'une ligne ou d'une colonne de neurones sur la couche compétitive selon les besoins de l'application de manière dynamique, au cours de la phase d'apprentissage.

La figure 4.8 présente un exemple de la procédure de l'apprentissage utilisée dans l'approche *GG-SOM*. Dans cette approche, une carte auto-organisatrice est initialement configurée selon les besoins de l'application en termes de dimension des vecteurs, paramètres de voisinage et paramètres d'apprentissage. La structure initiale de la carte, au démarrage, est minimale. Seulement quatre neurones distribués sur une grille 2×2 sont utilisés. Cette structure initiale effectue un ensemble d'itération d'apprentissage. A un instant T_1, les critères d'adaptation structurelle étant réunis, une opération d'incrémentation pourra être effectuée. Dans notre approche, l'incrémentation de la grille est lancée lorsqu'un neurone $n_{l,k}$ atteint un seuil de sélection $S_{s-l,k}$ défini par la taille de la carte ainsi que la dimension du vecteur de poids comme illustré par l'équation 4.2 :

$$S_s = (L \times K) \times D \tag{4.2}$$

avec *L* et *K* étant respectivement le nombre de lignes et colonnes de la structure actuelle et *D* étant la dimension des vecteurs d'entrée/poids.

Par contre, pour éviter l'ajout de neurones supplémentaires ne permettant pas d'augmenter la précision globale de l'application, un autre critère d'adaptation structurelle a été utilisé. Ce critère consiste à calculer l'erreur de quantification moyenne locale $EQM_{l,k}$ du neurone fréquemment sélectionné. Pour commencer l'incrémentation, il faut vérifier que $EQM_{l,k}$ a dépassé un seuil de précision P_s, le seuil qui a été fixé parmi les paramètres d'apprentissage de l'application. De cette façon, on évite qu'une zone dense dans l'espace d'entrée soit représentée par plusieurs neurones dont les vecteurs de poids sont relativement similaires. Par conséquent, chaque zone sera représentée par un nombre limité de neurones qui regrouperont tous ses échantillons.

Au cours d'une opération d'incrémentation, une *analyse statistique* sera effectuée, afin de décider sur la forme d'ensemble de neurones qui seront ajoutés (ligne ou colonne) ainsi que la position d'insertion de cet ensemble. En effet, l'insertion doit être effectuée entre le neurone le plus sélectionné et son voisin le plus éloigné en termes de vecteurs de poids. Suivant la décision

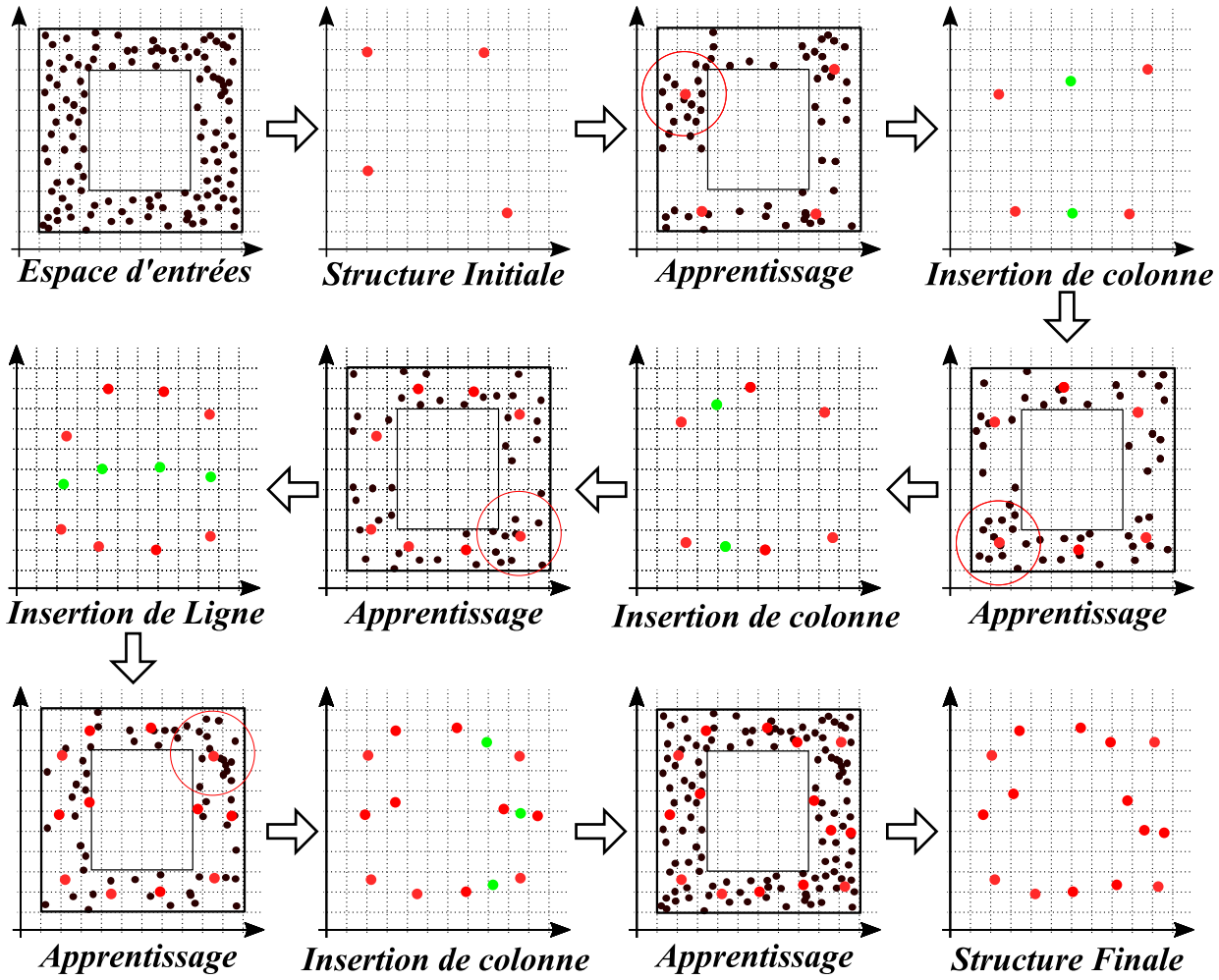


FIGURE 4.8 – *Procédure d'apprentissage avec l'approche Growing Grid SOM*

prise après cette analyse, un ensemble de neurones sera ajouté à la structure initiale. Par conséquent, les neurones appartenant à la ligne ou la colonne associée au neurone sélectionné seront clonés, avec une adaptation initiale de leurs vecteurs de poids pour qu'ils soient positionnés entre leurs voisins, notamment en termes de leurs valeurs de poids dans la carte *SOM*. De cette façon, la structure de la carte sera adaptée en gardant une *topologie régulière 2D maillée*.

A l'instant T_2 ($\mathrm{T}_2 > \mathrm{T}_1$), la nouvelle structure entre dans un nouveau cycle d'apprentissage classique. Un certain nombre d'itération d'apprentissage sera effectué avant d'atteindre le critère d'incrémentation ($S_s = (L \times K) \times D$ et $EQM_{l,k} > P_s$), à l'instant T_3 ($\mathrm{T}_3 > \mathrm{T}_2$). A ce moment, une nouvelle opération d'incrémentation sera effectuée. Cette alternance entre les opérations d'incrémentation et d'apprentissage classique se suit jusqu'à atteindre une structure finale, qui sera finalement utilisée au cours de la phase de rappel.

Afin d'empêcher la structure de la carte *SOM* de croître indéfiniment, un critère d'arrêt de l'apprentissage et de l'incrémentation de la grille doit être défini. Dans la littérature [100], quelques approches ont été proposées pour ce critère d'arrêt :

— **La taille maximale de la carte :** On fixe la taille maximale de la grille. Dans ce cas, l'incrémentation du nombre de lignes ou de colonnes s'arrête dès qu'on atteint la taille

maximale fixée de la carte *SOM*.

— **Le nombre de sélection** : le nombre de sélection des neurones doit être bien réparti. Dans ce cas, l'idée est d'ajouter des neurones pour éviter la surcharge éventuelle d'un neurone ou d'un groupe de neurones. Donc, les neurones fréquemment sélectionnés seront clonés et ces nouveaux neurones seront rajoutés à leur proximité pour répartir le nombre de sélections dans les itérations suivantes. Par conséquent, les échantillons représentant la totalité de l'espace d'entrée seront également répartis uniformément sur tous les neurones de la structure finale de la carte (critère dépendant de la qualité de projection de l'environnement externe).

— Éventuellement d'autres critères peuvent également être proposés en tenant particulièrement compte de la topologie de l'espace d'entrée. Dans ce cas, l'objectif est d'éviter les erreurs de quantification significatives, tout en minimisant le temps de traitement global (critère dépendant des caractéristiques de l'environnement externe).

Dans notre architecture, la taille maximale de la carte *SOM* est fixée par le nombre de neurones pouvant physiquement être implantés sur un support physique choisi. D'autre part, on cherche à augmenter les performances du système sans influencer la qualité des résultats obtenus. Dans ce cadre, deux critères d'arrêt ont été utilisés dans notre approche :

— **Taille maximale de la carte *SOM***. Ce critère est imposé par le nombre de neurones implantés. Si la taille maximale est atteinte, on arrête la procédure d'incrémentation et on procède à l'apprentissage classique pour diminuer l'erreur de quantification de la carte *SOM*.

— **Erreur de quantification globale**. Ce critère dépend de la qualité de quantification des caractéristiques de l'environnement externe projeté sur la structure de la carte *SOM* implémentée. Donc on fixe un seuil de précision P_s qui dépend de la tolérance d'erreur de l'application. Si l'erreur de quantification moyenne EQM_g est inférieure à ce seuil, on arrête l'apprentissage avec la structure actuelle de la carte *SOM*.

ii) Architecture matérielle de l'approche GG-SOM

En profitant de la flexibilité, de l'extensibilité et de l'adaptabilité de l'architecture *SOM-NoC*, nous avons réussi à proposer une architecture matérielle originale d'une carte auto-organisatrice évolutive. Cette architecture auto-adaptative permet de réaliser la procédure d'apprentissage détaillée précédemment d'une façon autonome. Comme présenté en figure 4.9, l'architecture *GG-SOM* est composée d'une carte *SOM* adaptable basée sur l'approche *SOM-NoC* et contrôlée par une unité de contrôle d'adaptation nommée « *Growing Grid Management* (*GCM*) ». Tout échange de données entre le *SOM-NoC* et le *GCM* ou l'extérieur du système est assuré par un gestionnaire d'entrées/sorties nommé « *Input/Output Management Circuit* (*IOMC*) ».

Les messages de l'architecture *SOM-NoC* (en entrée ou en sortie) sont gérés par le module *IOMC* qui assure le *paquetage* et/ou le *dépaquetage* des données sous une forme appropriée au réseau de communication *NoC*. Pour distribuer les données d'entrée vers tous les neurones de la

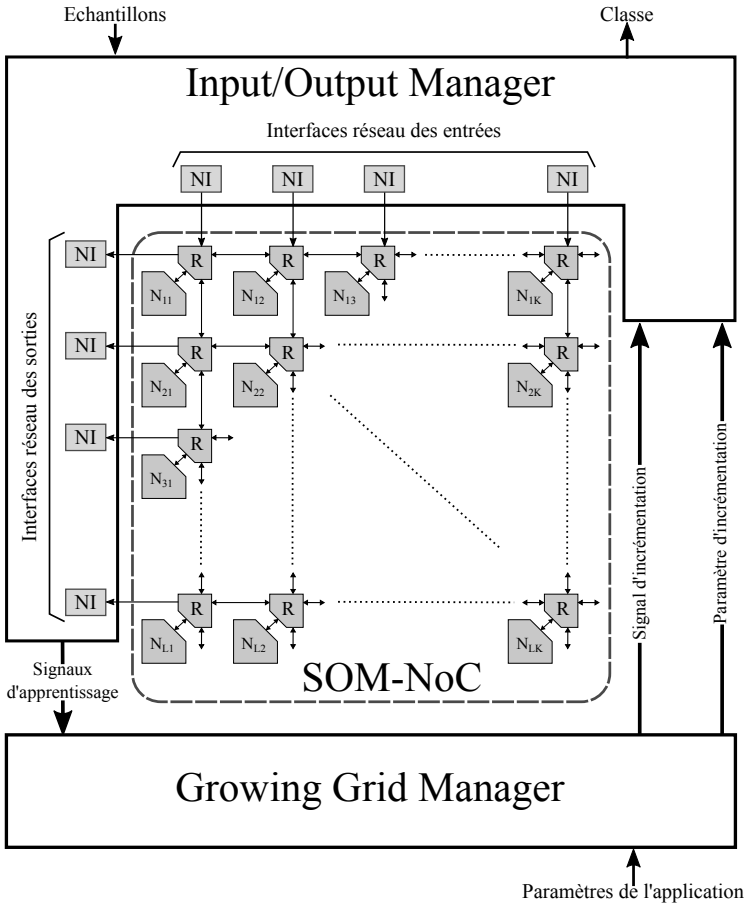


FIGURE *4.9 – Architecture de l'architecture GG-SOM*

carte *SOM*, ce module est connecté aux bus du côté « *Nord* » des routeurs situés à l'extrémité supérieure du *NoC*. Par contre, les données de sortie sont reçues sur les bus situés du côté « *ouest* » des routeurs placés du côté gauche du *NoC*. Dans les deux sens, tous les bus inoccupés sont connectés pour assurer la distribution parallèle par colonnes de données en entrée et la récupération parallèle par lignes des données en sortie. De l'autre côté, le module *IOMC* est connecté aux entrées/sorties du système et au module *GCM* afin d'envoyer les informations nécessaires à l'analyse statistique et de recevoir les ordres et les paramètres d'incrémentation de ce dernier.

Dans l'architecture *SOM-NoC*, les neurones sont distribués sur une topologie 2D maillée, où un neurone est composé d'un couple *routeur/ PE*. Dans l'architecture *GG-SOM*, chaque nœud est indexé par un couple d'adresse : « *adresse physique* » et « *adresse virtuelle* ». L'adresse physique est utilisée pour les opérations de communication. Cette adresse représente la position d'un nœud sur la grille des routeurs. Pour envoyer un message à un neurone, son adresse physique sera utilisée. Par conséquent, au cours du paquetage d'un message, au niveau des interfaces *Network Interface* (*NI*), l'adresse physique sera insérée dans la zone réservée à l'adresse du destinataire dans le *flit d'en-tête (header)*.

D'un autre côté, l'adresse virtuelle est utilisée pour les opérations de traitement (calcul) et d'incrémentation. Cette adresse permet d'identifier la position d'un neurone sur la couche compé-

titive de la carte auto-organisatrice. De plus, elle définit également le voisinage entre les neurones appartenant à la structure actuelle de la carte *SOM* utilisée à un moment donné.

Dans cette approche, on peut distinguer deux distributions : « *distribution physique* » et « *distribution virtuelle* ». Les opérations de traitement et d'incrémentation telles que le calcul de la distance, le calcul du taux de voisinage, l'analyse statistique et les ordres d'incrémentation sont effectuées suivant la distribution virtuelle. Alors que les opérations de communication telles que la distribution du vecteur d'entrée, la propagation systolique de résultats de comparaison et la rétro-propagation de l'identité du neurone gagnant sont effectuées suivant la distribution physique. Pour cela, une correspondance entre les deux adresses est effectuée au niveau des modules assurant le passage d'une opération de traitement à une opération de communication. Ainsi, au niveau de son interface réseau, chaque neurone reconnaît son couple d'adresses à base desquelles il pourra extraire les adresses de ses voisins les plus proches. D'un autre côté, au niveau du module *IOM* la correspondance des adresses physiques et virtuelles est effectuée à base d'une table de correspondance.

Le nombre de neurones physiques dans la grille *SOM* est fixé au cours de la conception. De plus, en profitant de l'évolutivité de l'approche *SOM-NoC* sur laquelle l'architecture *GG-SOM* s'appuie, la taille de la grille peut être augmentée, en combinant plusieurs modules *SOM*, comme présenté dans le chapitre précédent. De manière générale, la taille physique maximale de la carte auto-organisatrice sera fixée avant le démarrage du système.

Dans cette approche, un neurone physiquement implanté peut être dans un des deux états : « *actif* » ou « *inactif* ». A un instant *T*, les neurones participant à la structure actuelle de la carte *SOM*, qui est nommée la structure virtuelle, sont dans l'état « *actif* ». Le reste des neurones sont dans l'état « *inactif* » en attendant une activation éventuelle selon les besoins de l'application au cours du fonctionnement du système. Ces neurones ne participeront à l'application que lorsqu'ils seront invités à rejoindre la carte *SOM* par une opération d'incrémentation. Dans ce cas, une ligne ou une colonne de neurones inactifs les plus proches des neurones physiques de la carte *SOM* fonctionnelle sera activée.

Dans la distribution physique, les neurones récemment activés conserveront leurs positions initiales. Par contre, du point de vue de la structure virtuelle, ils seront placés dans une position permettant de séparer le voisinage virtuel entre le neurone le plus sélectionné comme gagnant et son voisin adjacent le plus éloigné en termes de distance entre leurs vecteurs de poids. Par conséquent, les correspondances des couples d'adresses seront mises à jour en fonction de la nouvelle distribution virtuelle. D'autre part, les vecteurs de poids des neurones nouvellement activés seront initialisés par la moyenne des vecteurs de poids des voisins virtuels adjacents, comme présenté par l'équation 4.3. A la fin de la phase d'apprentissage, la structure finale de la carte auto-organisatrice sera composée d'un ensemble de neurones activés.

$$W_{l,k} = \begin{cases} \frac{W_{l-1,k} + W_{l+1,k}}{2} & \text{si l'ensemble ajouté est une ligne} \\ \\ \frac{W_{l,K-1} + W_{l,k+1}}{2} & \text{si l'ensemble ajouté est une colonne} \end{cases} \tag{4.3}$$

avec (l, k) étant l'adresse virtuelle d'un neurone.

Un exemple de la procédure d'incrémentation de la structure de la carte *SOM* est illustré sur la figure 4.10. Dans cet exemple, la base d'apprentissage de l'espace d'entrée est constituée de 9 échantillons : *A, B, C, D, E, F, G, H et I*. Dans cet exemple, une carte auto-organisatrice composée de 3×3 neurones permettra de projeter de manière convenable l'espace d'entrée utilisé (9 valeurs). Au démarrage de la phase d'apprentissage, la *structure initiale de la carte SOM virtuelle* est composée de 2×2 *neurones*. Seuls les neurones situés à l'intersection des lignes *1* et *2* avec les colonnes *1* et *2* sont dans l'état *actif*. Les autres neurones sont initialisés dans l'état *inactif*. Dans cette structure initiale, les adresses virtuelles sont identiques aux adresses physiques.

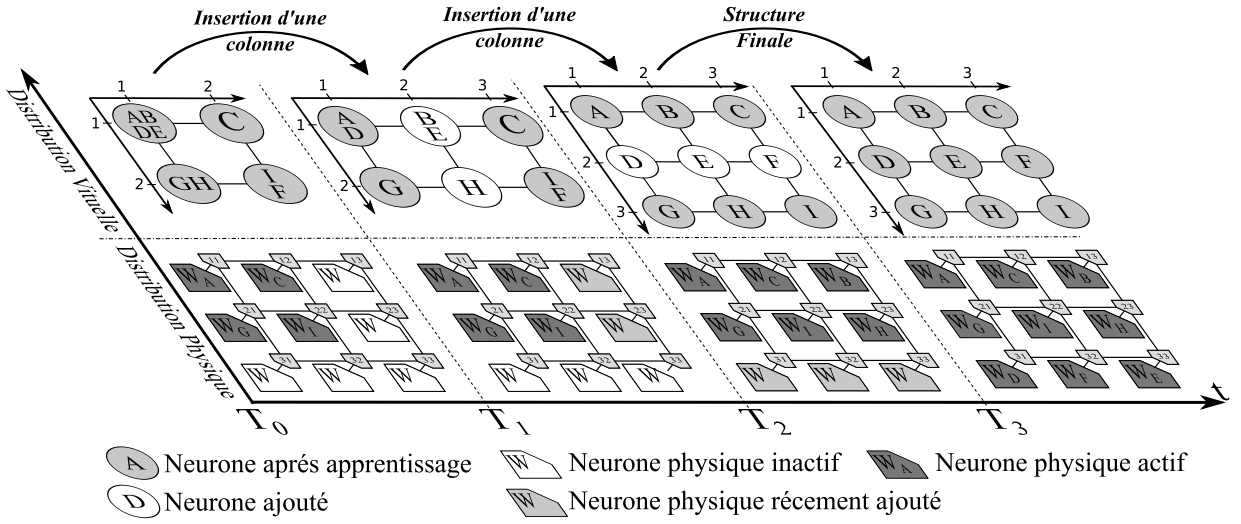


FIGURE *4.10 – Déroulement de l'incrémentation de la grille dans l'architecture GG-SOM*

À l'instant T_1, le neurone regroupant les échantillons « *A, B, D et E* » dépasse le seuil de sélection $\mathrm{S_s}$ avec une erreur élevée causée principalement par sa désignation pour des échantillons d'entrée ayant des valeurs différentes et séparées. Après une analyse statistique, le module *GCM* ordonne une opération d'incrémentation afin d'ajouter une nouvelle colonne. Cette opération consiste en l'activation des neurones situés, du point de vue de la structure physique, à l'intersection des lignes 1 et 2 avec la colonne 3 (la plus proche des neurones actifs). Ces neurones seront activés avec un pré-apprentissage selon l'équation 4.3. Entre autre, la structure de la carte *SOM* change après cette opération. Du point de vue de la structure virtuelle, les neurones ajoutés seront placés dans la colonne 2 de la grille de la carte *SOM* virtuelle. Par conséquent, avec la nouvelle structure, les adresses virtuelles seront différentes des adresses physiques. Ainsi, les communications seront gérées par le *NoC* via la correspondance de couple d'adresses physique et virtuelle. Ces correspondances seront mises à jour uniquement dans les neurones actifs ainsi

que au sein du module *IOMC*. En résumé, l'architecture *SOM-NoC* s'auto-adapte et entre dans un nouveau cycle d'apprentissage classique avec la nouvelle structure de la carte *SOM*, de taille 2×3.

À l'instant T_2, comme illustré à la figure 4.10, une autre opération d'incrémentation sera lancée, et cette fois après avoir dépassé le seuil de sélection S_s par l'un des neurones actifs. Dans l'exemple présenté, une ligne sera ajoutée de la même manière que précédemment. Enfin, à l'instant T_3 (voir la figure 4.10), la phase d'apprentissage sera achevée avec une structure optimale de la carte auto-organisatrice pour l'espace d'entrée utilisé. Du point de vue de la structure physique, la disposition des neurones dans cette carte physique est désordonnée. Par contre, on remarque que dans la distribution virtuelle, les disposition des neurones et la projection de l'espace d'entrée sont bien organisées. En effet, c'est la seule distribution qui compte pour l'exécution de l'application, puisque seulement les positions des neurones dans cette distribution virtuelle sont utilisées au cours la phase d'apprentissage ainsi que la phase de rappel.

iii) Module de gestion de l'incrémentation de l'architecture GG-SOM

Growing Grid Management (*GCM*) est le module responsable de la supervision de la phase d'apprentissage. Ce module collecte les informations au cours du processus d'apprentissage classique pour décider si la phase d'apprentissage doit être terminée et passer à la phase de rappel ou effectuer une opération d'incrémentation pour ajouter une ligne ou une colonne de neurones pour adapter structurellement la carte auto-organisatrice aux nouveaux besoins. Pour chaque nouvelle application, des paramètres tels que la dimension des vecteurs, le seuil de précision tolérée et la taille maximale imposée par les contraintes temporelles et qualitatives seront collectés par le module *GCM*. A base de ces paramètres et des échantillons d'entrée envoyés au cours de la phase d'apprentissage, le module *GCM* permet de modifier la structure de la carte *SOM* virtuelle pour qu'elle soit la mieux adaptée à l'application donnée évoluant dans un environnement d'entrée spécifique.

L'algorithme 1 décrit les opérations effectuées par le module *GCM* pendant la phase d'apprentissage. Suivant cet algorithme la supervision de l'apprentissage passe régulièrement par trois phases : la collecte des informations, l'analyse statistique et la décision.

Algorithme 1 : Gestion de l'apprentissage évolutif au niveau du *GGM*

Variables

$L^* \times K^*$	:	Taille maximale du SOM
$L \times K$	:	Taille actuelle du SOM
D	:	Dimension des vecteurs $\overrightarrow{X}$ et $\overrightarrow{W}$
P_s	:	Seuil de précision relatif à l'application
S_s	:	Seuil de sélection d'un neurone relatif à la structure virtuelle du SOM
L_P	:	Statut d'apprentissage (Classic, Growing et End)
$N_{l,k}$	:	Le neurone le plus sélectionné au cours de l'apprentissage classique
E_{GQ}	:	Erreur de quantification globale du SOM
E_{IQ}	:	Erreur de quantification d'une itération
N_I	:	Nombre d'itérations d'apprentissage classique
$E_{LQ_{l,k}}$	:	Erreur de quantification locale du neurone $N_{l,k}$
$S_{l,k}$	:	Nombre de sélection du neurone $N_{l,k}$
$D_{(l,k):(i,j)}$	:	Distance entre les vecteur de poids $\overrightarrow{W}_{\mathrm{l,k}}$ du neurone $N_{l,k}$ et $\overrightarrow{W}_{\mathrm{i,j}}$ de son voisin $N_{i,j}$

Initialisation

$L \leftarrow 2$
$K \leftarrow 2$
$S_s \leftarrow L \times K \times D$
$E_{GQ} \leftarrow 0$
$N_I \leftarrow 0$
$L_P \leftarrow Classic$
Pour tout $N_{i,j}$ $(0 \leq i < L, 0 \leq j < K)$ **fais**
  $E_{LQ_{i,j}} \leftarrow 0$
  $S_{i,j} \leftarrow 0$
Fin pour

Traitement

Tant que $L_P \neq End$ **fais** *// Pour chaque itération d'apprentissage*
  Si $L_P = Classic$ **Alors**
    – Phase de collecte des informations
    Si $N_{l,k} = Winner$ **Alors**
      $E_{GQ} \leftarrow E_{GQ} + E_{IQ}$
      $N_I \leftarrow N_I + 1$
      $E_{LQ_{l,k}} \leftarrow E_{LQ_{l,k}} + E_{IQ}$
      $S_{l,k} \leftarrow S_{l,k} + 1$
      – Phase d'analyse statistique
      Si $S_{l,k} \geq S_s$ **Alors**
        Si $E_{GQ} \geq P_s \times N_I$ **Alors**
          $L_P \leftarrow End$
        Sinon $E_{LQ_{l,k}} \geq P_s \times S_{l,k}$ **Alors**

La suite sur la page suivante

(Suite) Algorithme 1 : Gestion de l'apprentissage évolutif au niveau du GGM

Requêtes des poids des neurones : $N_{l,k}$, $N_{l+1,k}$, $N_{l,k+1}$, $N_{l-1,k}$ et $N_{l,k-1}$
$L_P \leftarrow Growing$
Sinon
$L_P \leftarrow Classic$
Fin si
Fin si
Fin si
Sinon si $L_P = Growing$ **Alors**
Attendre les poids des neurones : $N_{l,k}$, $N_{l+1,k}$, $N_{l,k+1}$, $N_{l-1,k}$ et $N_{l,k-1}$
$D_{(l,k):(l,k+1)} \leftarrow \|N_{l,k+1} - N_{l,k}\|$
$D_{(l,k):(l,k-1)} \leftarrow \|N_{l,k-1} - N_{l,k}\|$
$D_{(l,k):(l+1,k)} \leftarrow \|N_{l+1,k} - N_{l,k}\|$
$D_{(l,k):(l-1,k)} \leftarrow \|N_{l-1,k} - N_{l,k}\|$
$D_{max} \leftarrow max(D_{(l,k):(l,k+1)}, D_{(l,k):(l,k-1)}, D_{(l,k):(l+1,k)}, D_{(l,k):(l-1,k)})$
– Phase de décision
Si $D_{(l,k):(l,k+1)} = D_{max}$ **Alors**
Ajouter une colonne ***C*** *@* ***k+1***
$K \leftarrow K + 1$
Sinon si $D_{(l,k):(l+1,k)} = D_{max}$ **Alors**
Ajouter une ligne ***R*** *@* ***l+1***
$L \leftarrow L + 1$
Sinon si $D_{(l,k):(l,k-1)} = D_{max}$ **Alors**
Ajouter une colonne ***C*** *@* ***k***
$K \leftarrow K + 1$
Sinon
Ajouter une ligne ***R*** *@* ***l***
$L \leftarrow L + 1$
Fin si
$S_s \leftarrow L \times K \times D$
$E_{GQ} \leftarrow 0$
$N_I \leftarrow 0$
$L_P \leftarrow Classic$
Pour tout $N_{i,j}$ $(0 \leq i < L, 0 \leq j < K)$ **fais**
$E_{LQ_{i,j}} \leftarrow 0$
$S_{i,j} \leftarrow 0$
Fin Pour
Fin si
Fin Tant que
FIN

- **La collecte des informations :** Pendant un cycle d'apprentissage classique, le module *GCM* reçoit l'adresse virtuelle du neurone gagnant $N_{l,k}$ et l'erreur de quantification (la distance entre le vecteur d'entrée et le vecteur de poids du neurone gagnant), pour chaque itération d'apprentissage. A base de ces informations, il calcule les paramètres suivants : le nombre

d'itérations d'apprentissage traitées N_I, l'accumulation de l'erreur de quantification globale E_{GQ}, le nombre de sélection $S_{l,k}$ du neurone gagnant et l'accumulation de son erreur de quantification locale $E_{LQ_{l,k}}$. Ces données seront utilisées au cours de la phase d'analyse statistique, pour décider d'arrêter l'apprentissage ou d'effectuer une éventuelle opération d'incrémentation.

• **L'analyse statistique :** Cette phase commence lorsque le nombre de sélections $S_{l,k}$ de l'un des neurones actifs dépasse le seuil de sélections S_s. Dans ce cas, si l'erreur de quantification globale E_{GQ} est inférieure au seuil de précision P_s relatif à l'application, le critère d'arrêt de l'apprentissage est vérifié. Par conséquent, l'apprentissage est achevé avec la structure virtuelle actuelle de la carte *SOM*. Cette structure sera ensuite utilisée au cours de la phase de rappel. Dans le cas contraire, une analyse de précision du neurone sélectionné $N_{l,k}$ sera effectuée pour vérifier le critère d'incrémentation. Entre autre, on vérifie si l'erreur de quantification moyenne locale de ce neurone est plus grande que le seuil de précision ($E_{LQ_{l,k}} > P_s$). Sinon, on continue l'apprentissage classique jusqu'à ce que l'un des deux critères, le critère d'arrêt ou d'incrémentation, sera vérifié. L'idée est d'éviter le clonage des neurones représentant une zone dense de l'espace d'entrée. Par contre, si l'erreur de quantification locale $E_{LQ_{l,k}}$ est grande, cela signifie que le neurone représente une zone large de l'espace d'entrée. Dans ce cas, on doit ajouter des neurones supplémentaires dans son voisinage afin de distribuer et mieux représenter cette projection de l'espace d'entrée. Par conséquent, une opération d'incrémentation s'avèrera nécessaire dans ce cas de figure.

  Au cours de la phase d'analyse statistique, l'architecture prépare les informations nécessaires pour la décision sur l'opération d'incrémentation. En premier lieu, une requête est envoyée au neurone sélectionné ainsi qu'à ces voisins virtuels adjacents afin de récupérer leurs vecteurs de poids. Une fois reçus, ces vecteurs de poids seront ensuite utilisés pour calculer les distances $D_{(l,k):(i,j)}$ entre le neurone sélectionné $N_{l,k}$ et ces voisins adjacents $N_{l,k+1}$, $N_{l+1,k}$, $N_{l,k-1}$ et $N_{l-1,k}$. La distance la plus grande sera utilisée comme paramètre de génération de l'ordre d'incrémentation, au cours de la phase de décision.

• **La décision :** Au cours de cette phase, le module *GCM* génère l'ordre d'incrémentation. Cet ordre est composé de deux informations : le type de l'ensemble de neurones à ajouter (ligne ou colonne) et la position d'ajout dans la nouvelle distribution virtuelle. En effet, l'ensemble ajouté doit être placé entre le neurone le plus sélectionné et son voisin le plus éloigné en termes de la distance $D_{(l,k):(i,j)}$ précédemment calculée. Ensuite, l'ordre d'incrémentation sera envoyé au module *IOMC*. Enfin, *GCM* initialise ses variables pour se préparer à un nouveau cycle d'apprentissage classique qui commencera dès que l'incrémentation sera effectuée au niveau de la carte *SOM-NoC*.

A la réception de l'ordre d'incrémentation de la carte, le module *IOMC* met à jour la table de correspondances entre les adresses physiques et virtuelles. Ensuite, il s'adapte à cette nouvelle structure en diffusant l'ordre d'incrémentation à tous les neurones de la carte actuelle pour que eux à leur tour effectuent les opérations nécessaires pour se préparer à cette nouvelle structure.

iv) Architecture d'un neurone GG-SOM

La principale différence d'un neurone de l'architecture *GG-SOM* par rapport à un neurone de l'approche *SOM-NoC* présenté dans le chapitre précédent est la présence de deux adresses : une *adresse virtuelle* utilisée pour les opérations de traitement et une *adresse physique* utilisée pour les opérations de communication. La correspondance de ces deux adresses au sein d'un neurone de l'architecture *GG-SOM* est effectuée dans l'interface *NI*, qui assure également la connexion avec le réseau sur puce et indirectement avec les autres neurones de l'architecture.

Un neurone de l'architecture *GG-SOM* est composée de 5 unités, comme le neurone *SOM-NoC* : *VEP*, *USG*, *LWS*, *NI* et *LCM*. Par rapport à l'architecture proposée dans l'approche de la carte *SOM-NoC* adaptable, seuls les modules *VEP*, *USG* et *LCM* ont été modifiés pour pouvoir assurer cette propriété d'évolutivité caractéristique pour les *SOM* évolutifs. Le module *USG* a été modifié au niveau de l'initialisation du rayon de voisinage *R(t)*. Pour chaque neurone activé dans une phase d'incrémentation, son rayon de voisinage sera initialisé à *1*. De cette façon, les neurones ne seront concernés par l'adaptation de leurs vecteurs de poids que s'ils sont sélectionnés comme gagnant ou éventuellement si un de leurs voisins adjacents est sélectionné comme neurone gagnant. De plus, une nouvelle opération de traitement est ajoutée au module *VEP*. Cette opération consiste à calculer les valeurs initiales des éléments du vecteur de poids d'un neurone récemment activé à la fin de la procédure d'incrémentation, selon l'équation 4.3.

Le module *LCM* est l'unité responsable de l'adaptabilité des neurones de l'architecture *GG-SOM*. Ce module possède toutes les informations de la structure virtuelle de la carte *SOM* (dimension de la carte *SOM*, taille du vecteur d'entrée, positions virtuelle et physique). Son rôle est d'envoyer des ordres aux différents modules du neurone pour adapter dynamiquement leurs comportements en cas de changement de structure. Dans l'architecture *GG-SOM*, le module *LCM* est également responsable de l'activation des neurones et de la gestion de la procédure d'incrémentation au niveau d'un neurone, à la réception d'un ordre d'incrémentation.

Toutes les opérations effectuées au sein d'un neurone sont séquencées par une machine à états finis, illustrée à la figure 4.11. Cette machine à états finis est composée de 6 états : *Idle*, *Init*, *Distance*, *Update*, *Send weight* et *Growing*. Le changement d'un état à un autre est principalement effectué à la réception des messages au niveau du *décodeur de l'interface réseau* NI.

- **Idle –** C'est un état de veille. Le module *LCM* est positionné dans cet état en attendant la réception d'un message de données ou de configuration.
- **Init –** A la réception d'un message d'initialisation, le module LCM d'un neurone de la structure initiale de la carte *SOM* configure son *VEP*, en fixant le nombre d'éléments du vecteur poids suivant la dimension du vecteur d'entrée. En outre, il initialise les paramètres de mise à jour dans l'unité *USG*. Enfin, il initialise l'adresse virtuelle du neurone, qui sera égale à l'adresse physique, et met le neurone dans l'état actif.
- **Distance -** A la réception du premier élément du vecteur $\overrightarrow{X}$, le module *LCM* ordonne au *VEP* de commencer le calcul de la distance Euclidienne entre le vecteur de poids et le vecteur

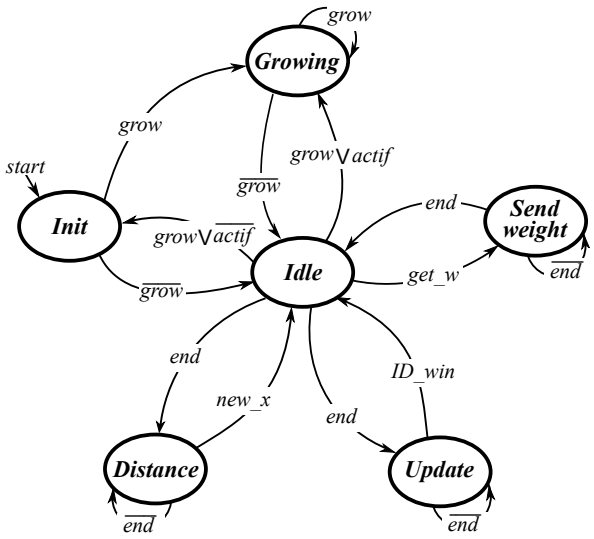


FIGURE *4.11 – Machine à états finis du module de configuration locale LCM*

d'entrée, dont le reste de ses éléments seront reçus de manière séquentielle via le *NoC*.

• **Update -** L'état de mise à jour sera activé à la fin d'une itération d'apprentissage. L'identité virtuelle du neurone gagnant sera transmise à tous les neurones actifs. Lors de la réception de cette identité virtuelle, le module *LCM* demande au module *USG* de générer les signaux de mise à jour et au module *VEP* de procéder à l'adaptation des éléments du vecteur de poids.

• **Send weight –** Au cours de l'étape d'analyse statistique au niveau du module *GCM*, une demande de récupération des poids sera envoyée au neurone gagnant et à ses voisins les plus proches dans la carte virtuelle. A la réception de cette requête, le module LCM ordonne au *VEP* d'envoyer les éléments du vecteur de poids vers le codeur de l'interface réseau qui les transmettra à son tour vers le module *IOMC* via le *NoC*. Enfin, les vecteurs de poids du neurone sélectionné et ses voisins virtuels seront récupérés par le module *GCM*.

• **Growing –** A la réception de l'ordre d'incrémentation, les neurones qui seront ajoutés dans la carte virtuelle passeront obligatoirement par l'état *Init* pour préparer leurs modules constitutifs. Une fois cette préparation terminée, ils passeront ensuite à l'état *Growing* dès la réception de l'ordre d'incrémentation. Cet état est un état de reconfiguration. Les informations sur la structure virtuelle de la carte *SOM* seront mises à jour. La taille de la carte *SOM* sera changée avec une incrémentation du nombre de lignes L si l'ordre consiste à ajouter une ligne, sinon le nombre de colonnes K sera incrémenté. De plus, l'*adresse virtuelle* sera mise à jour en fonction de la position de l'ensemble de neurones ajouté, comme présenté par l'algorithme 2. Ensuite, un ordre sera envoyé au *VEP* pour procéder à l'initialisation des poids des neurones ajoutés.

Algorithme 2 : Gestion de l'incrémentation au niveau du *LCM*

Variables

$L \times K$: *Taille actuelle de la carte SOM*
(x, y) : *Adresse virtuelle du neurone*
(x^*, y^*) : *Adresse physique du neurone*
ADD : *Type de l'ensemble ajouté C pour colonne et R pour ligne*
P : *Position de l'ajout de l'ensemble sur la grille virtuelle*

Initialisation

$L \leftarrow 2$
$K \leftarrow 2$
$x \leftarrow x^*$
$y \leftarrow y^*$

Traitement

Pour chaque *réception (ADD, P)* **fait** – *A la réception d'un ordre d'incrémentation*
  Si $ADD = C$ **Alors** – *Ajouter une colonne*
    $K \leftarrow K + 1$
    – Mise à jour de l'adresse virtuelle
    Si $y = K$ **Alors**
      $y \leftarrow P$
    Sinon si $y \geq P$ **Alors**
      $y \leftarrow y + 1$
    Fin si
    – Activation et initialisation des neurones ajoutés
    Si $y = P$ **Alors**
      Activer et configurer le neurone
      Attendre les poids des neurones : $N_{x,P+1}$ *et* $N_{x,P-1}$
      $W_{x,y} \leftarrow \frac{W_{x,P-1} + W_{x,P+1}}{2}$
    Sinon si $y = P + 1$ *ou* $y = P - 1$ **Alors**
      Envoyer ses poids au neurone : $N_{x^*,K}$
    Fin si
  Sinon – *Ajouter une Ligne*
    $L \leftarrow L + 1$
    Si $x = L$ **Alors**
      $x \leftarrow P$
    Sinon si $x \geq P$**Alors**
      $x \leftarrow x + 1$
    Fin si
    Si $x = P$ **Alors**
      Activer et configurer le neurone
      Attendre les poids des neurones : $N_{P+1,y}$ *et* $N_{P-1,y}$
      $W_{x,y} \leftarrow \frac{W_{P-1,y} + W_{P+1,y}}{2}$

La suite sur la page suivante

(Suite) Algorithme 2 : Gestion de l'incrémentation au niveau du LCM

Sinon si $x = P + 1$ *ou* $x = P - 1$ **Alors**
Envoyer ses poids au neurone : N_{L,y^*}
Fin si
Fin si
Fin Pour
FIN

Le traitement local de l'opération d'incrémentation, comme illustré avec l'algorithme 2, dépend de l'adresse virtuelle du neurone ainsi que de la position virtuelle de l'ensemble de neurones ajoutés dans la structure virtuelle initiale. Si un neurone est situé au voisinage virtuel direct de la nouvelle ligne ajoutée, son module *LCM* envoie son vecteur de poids au neurone le plus proche dans cette nouvelle structure virtuelle appartenant à cette nouvelle ligne. Si la nouvelle ligne de neurones est ajoutée dans la carte virtuelle entre deux lignes de neurones existantes, chaque neurone de la ligne nouvellement ajoutée aura deux neurones voisins virtuels qui lui enverront leurs propres vecteurs de poids. Ces vecteurs de poids sont ensuite utilisés pour faire une moyenne et initialiser les poids des nouveaux neurones (voir l'équation 4.3). La même procédure est effectuée dans le cas de figure où une colonne est ajoutée à la structure initiale. Dans le cas contraire, si le neurone est situé sur la ligne (ou colonne) ajoutée, il doit attendre la réception des éléments de poids de ses voisins virtuels pour initialiser son vecteur de poids selon l'équation 4.3. Dans le cas contraire, le neurone ne participe pas à la procédure d'incrémentation et de croissance de la carte et son module *LCM* passe à l'état de veille après la mise à jour de son adresse virtuelle.

4.3.3 Résultats expérimentaux

L'architecture matérielle *GG-SOM* proposée a été décrite avec un code *VHDL* générique et synthétisée pour une implantation sur carte *FPGA* de type *Virtex-7 Xilinx VC707*. Cette description permet de modifier la structure de la carte *SOM* physique implantée par un simple paramétrage en modifiant la dimension *D* du vecteur d'entrée ainsi que le nombre de neurones en fonction du nombre de lignes *L* et du nombre de colonne *K*.

Plusieurs structures *SOM* ont été synthétisées en utilisant *Xilinx ISE Design Suite 14.7*. Les ressources matérielles utilisées avec cette architecture augmentent linéairement en augmentant la taille physique de la carte *SOM* implantée. Par contre, on note que la fréquence maximale obtenue, pour toute architecture implémentée, est aux alentours de *250 MHz*. La tableau 4.3 présente les résultats de synthèse de l'architecture *GG-SOM* de taille maximale 10×10. La partie des ressources utilisées par chaque module de l'architecture, comme le montre la figure 4.9, est également représentée dans ce tableau. On remarque que les modules de contrôle (*IOMC* et *GCM*) ne nécessitent pas beaucoup de ressources par rapport à celles utilisées par les modules *SOM-NoC*.

En revanche, bien qu'ils offrent une flexibilité et une évolutivité à l'architecture proposée,

TABLEAU *4.3 – Résultat de synthèse de l'architecture GG-SOM sur une carte FPGA VIRTEX-7 XC7VX485T*

Ressources	**GG-SOM** 10×10	**SOM-NoC**		**IOMC**	**GGM**
		Neurone	**Routeurs**		
Registres	*255 654*	*56 %*	*39.3 %*	*1 %*	*3.7 %*
LUT	*267 979*	*48.3 %*	*48.2 %*	*1 %*	*2.5 %*
DSP	*104*	*96.2 %*	*0 %*	*0 %*	*3.8 %*
BRAM	*102*	*98.1 %*	*0 %*	*0 %*	*1.9 %*

les modules de communication (routeurs) consomment une partie non négligeable des ressources matérielles. Cela est essentiellement dû à l'utilisation d'un réseau sur puce *NoC classique*. Cet inconvénient pourrait être corrigé en utilisant des *NoC* plus adaptés aux opérations de communication au sein de l'architecture *GG-SOM* et pourrait également offrir de meilleures performances en termes de MCPS/MCUPS de cette architecture proposée. Dans la suite de ces travaux de recherche, ces approches de communication de type NoC adaptées aux spécificités d'une carte *SOM* ne seront pas abordées davantage.

Le tableau 4.4 présente les paramètres de l'architecture *GG-SOM*. Les performances temporelles de l'architecture *GG-SOM* ont été estimées à base de ces paramètres. Il y a 6 opérations différentes lors du fonctionnement de la carte *GG-SOM* : les opérations de communication et les opération de traitement. Les opérations de communication regroupent *la distribution des échantillons d'entrée vers tous les neurones, la propagation systolique et la retro-propagation de l'identité du neurone gagnant*. Le temps d'exécution de ces opérations est calculé en fonction de la taille de la grille virtuelle de la carte *SOM* ($L \times K$). D'un autre côté, les opérations de traitement regroupent *le calcul de la distance et l'adaptation des poids*, dont les temps d'exécution dépendent de la dimension du vecteur (D). Enfin, l'opération d'incrémentation est basée sur des opérations de communication ainsi que des opérations de traitement et de reconfiguration. Le temps d'exécution de cette opération dépend de l'ancienne structure de la carte *SOM*, au cours de l'analyse statistique et de la nouvelle structure de la carte *SOM*, au cours de la phase d'incrémentation. De surcroit, le temps d'exécution de cette opération d'incrémentation dépend également de la dimension du vecteur D.

Les performances temporelles de l'architecture *GG-SOM* sont présentées dans le tableau 4.5. Ces performances sont exprimées avec des expressions mathématiques permettant d'extraire le temps d'exécution de chaque opération pendant *la phase d'apprentissage* ainsi que *la phase de rappel*. On remarque que le temps d'exécution, pour une application donnée avec un vecteur de dimension fixe, augmente avec le nombre de neurones actifs sur la grille. Pour cela, il est préférable d'un point de vue des performances, d'utiliser un nombre minimal de neurones permettant de garantir des résultats satisfaisants en termes de précision. Pour atteindre cet objectif, il faut arrêter la phase d'apprentissage dès que la moyenne de l'erreur de quantification globale du système atteint une valeur inférieure à l'erreur du seuil de précision toléré par l'application.

TABLEAU *4.4 – Paramètres de l'architecture GG-SOM implantée*

Paramètre	**Valeur**	**Description**
$L \times K$	$L \times K = 10 \times 10$	Taille du SOM
D	$D = 3$	Dimension de vecteurs d'entrée/poids
N_{es}	$N_{es} = L + K - 1 = 19$	Nombre d'étages systoliques
$N_{v,(i,j)}$	$N_{v,(i,j)} = \begin{cases} 2 & si\ i = (1,L)\ et\ j = (1,K) \\ 3 & si\ 1 < i < L\ et\ j = (1,K) \\ 3 & si\ i = (1,L)\ et\ 1 < j < K \\ 4 & si\ 1 < i < L\ et\ 1 < j < K \end{cases}$	Nombre de voisins concernés par l'analyse statistique pour un neurone $N_{(i,j)}$
T_r	$T_r = 2$	Latence d'un routeur
T_c	$T_c = 1$	Temps de paquetage
T_d	$T_d = 1$	Temps de dépaquetage

TABLEAU *4.5 – Performances temporelles de l'architecture GG-SOM*

Type	**Opération**	**Formule**
Communication	Distribution du vecteur $\overrightarrow{X}$	$\mathrm{T_{is}} = T_c + T_d + T_r \times L$
	Propagation systolique	$T_P = (T_c + 2 \cdot T_r + 2 \cdot T_d) \times (N_{es} - 1)$
	Retro-propagation de l'identité du neurone gagnant	$T_{Rp} = 2 \cdot T_c + 2 \cdot T_d + T_r \times (L + K)$
Traitement	Calcul de la distance	$T_{cd} = D + 3$
	Adaptation des poids	$T_{upd} = D + 1$
Incrémentation	Growing	$T_{grow} = \begin{cases} T_{sel} + T_{add\_R} & \text{si l'ensemble ajouté est une ligne} \\ T_{sel} + T_{add\_C} & \text{si l'ensemble ajouté est une colonne} \end{cases}$
	Analyse statistique	$T_{sel} = \left[N_{v,(i,j)} + 1\right] \times [D + 2 \cdot T_c + 2 \cdot T_d + T_r \times (L + K)]$
	Ajout d'une ligne	$T_{add\_R} = 2 \cdot (Dim + 1) + 2 \cdot T_c + 2 \cdot T_d + T_r \times (L + 1)$
	Ajout d'une colonne	$T_{add\_C} = 2 \cdot (Dim + 1) + 2 \cdot T_c + 2 \times T_d + T_r \times (L + K)$

D'autre part, l'apprentissage évolutif basé sur l'incrémentation de la taille de la grille de la couche compétitive, permet de réduire le temps d'exécution de la phase d'apprentissage, en commençant par une petite carte qui sera incrémentée au cours du processus d'apprentissage. De cette façon, le temps d'exécution des premières itérations d'apprentissage est inférieur à celui d'une carte auto-organisatrice statique, même si la structure finale est de taille similaire voire identique. De surcroit, cet avantage de performance en faveur de l'approche *GG-SOM* n'affecte pas la qualité des résultats obtenus. Dans ce cadre, l'architecture *GG-SOM* a également été validée sur un exemple de système de compression d'images. Les résultats de cette validation ont été comparés avec ceux obtenus avec le système de compression d'images à base de l'architecture *SOM-NoC statique* présentée dans la section 3.7. Ces résultats sont détaillés dans la section suivante.

4.3.4 Validation de l'architecture GG-SOM sur une application de quantification d'image

Un système de compression d'images a été conçu pour la validation de l'architecture *GG-SOM*. Ce système accepte en entrée les pixels d'une image *RGB*. Ainsi, chaque échantillon d'entrée de cette application est considéré comme un vecteur de trois éléments. Les expériences ont été réalisées sur des images *RGB* de résolution 128×128 *pixels*.

Afin de comparer le système de compression d'image à base de l'architecture *GG-SOM* au système de compression à base de l'architecture *SOM-NoC statique* de taille 10×10 (proposée et détaillée dans la section 3.7), nous avons fixé la taille maximale de *GG-SOM* à 10×10. Quatre image, illustrées à la figure 4.13, ont été utilisées : *Lenna, Airplane, Pepper* et *Parrot*. Pour commencer, nous avons utilisé le système à base de *SOM-NoC statique* pour extraire les résultats qualitatifs obtenus après *16 384 itérations d'apprentissage* pour chaque image. Ces résultats ont été utilisés pour calculer le paramètre P_s représentant le *seuil de précision* nécessaire comme un critère d'arrêt pour le système à base de *GG-SOM*.

Afin de comparer la précision des systèmes en termes de la moyenne de l'erreur globale de quantification ainsi que le temps d'exécution de la phase d'apprentissage sur chaque système, nous avons effectué la compression des quatre images sur les deux systèmes, à base de *GG-SOM* et de *SOM-NoC statique* de taille 10×10 chacun, avec la même base d'apprentissage de *25 000 échantillons* d'entrée. La figure 4.12 présente l'évolution de la moyenne de l'erreur de quantification globale pour les deux systèmes en fonction du nombre d'itérations d'apprentissage (nombre d'échantillons d'entrée) pour les quatre images (*Lenna, Airplane, Pepper* et *Parrot*). D'autre part, le temps d'exécution de la phase d'apprentissage en fonction du nombre d'itérations est présenté sur une échelle logarithmique.

On remarque qu'au début de la phase d'apprentissage, l'erreur de quantification du système basé sur le *SOM-NoC statique* est inférieure à celle du système basé sur l'approche *GG-SOM*. Cela est dû au nombre de neurones utilisés sur la couche compétitive. Ainsi, le nombre de couleurs sur *la palette GG-SOM* est petit et moins diversifié, tandis que le temps d'exécution est inférieur. Ceci peut être expliqué en se référant aux expressions présentées dans le tableau 4.5. Avec l'incrémentation de la grille et par conséquent du nombre de neurones dans l'architecture *GG-SOM*, l'erreur de quantification commence à diminuer essentiellement à cause de l'augmentation de la variété de couleurs sur la palette. L'erreur de quantification diminue également dans le système *SOM-NoC statique* en fonction du nombre d'itérations d'apprentissage, mais avec un taux de convergence inférieur à celui de l'architecture *GG-SOM*. D'autre part, le temps d'exécution pour l'apprentissage reste inférieur pour le système à base de l'architecture *GG-SOM*.

Après plusieurs itérations d'apprentissage, la moyenne de l'erreur globale de quantification du système *GG-SOM* devient plus petite que la précision minimale souhaitée (définie par le seul $\mathrm{P_S}$). Au moment où le système atteint cette précision demandée, la phase d'apprentissage du système s'arrête. On remarque que le nombre d'itérations nécessaires pour atteindre la précision souhaitée

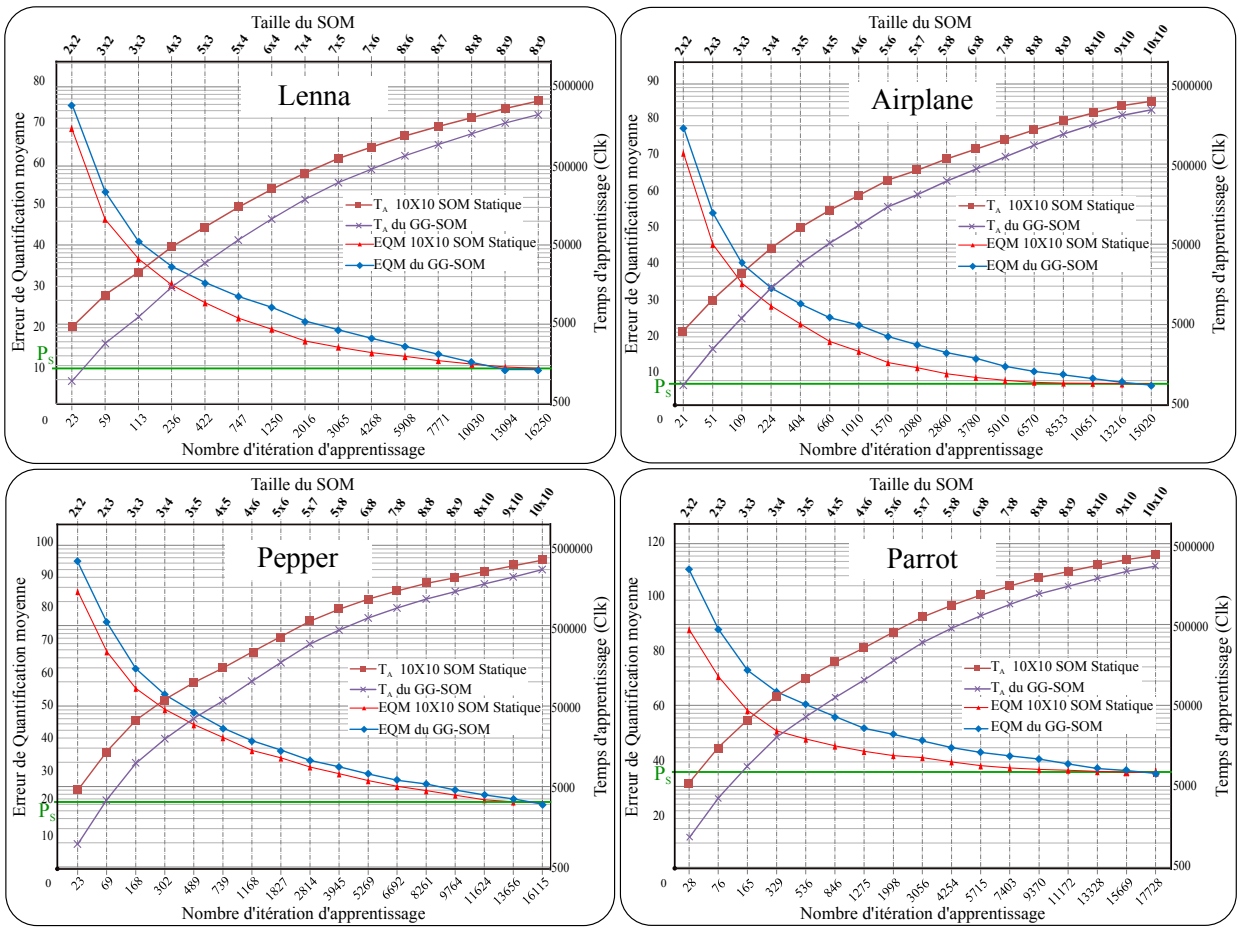


FIGURE *4.12 – Variation de l'erreur de quantification et du temps d'apprentissage en fonction du nombre d'itérations d'apprentissage*

TABLEAU *4.6 – Comparaison des résultats d'apprentissage obtenus par un système de compression à base de l'architecture GG-SOM et un système de compression à base de l'architecture SOM statique*

Image	**Nombre d'itération**		**Temps d'apprentissage** (clk)		$\mathbf{EQM_g}$	
	GG-SOM	**SOM-NoC**	**GG-SOM**	**SOM-NoC**	**GG-SOM**	**SOM-NoC**
Lenna	*13 094*	*16 205*	*1 904 044*	*3 273 410*	*9.10*	*9.49*
Airplane	*15 020*	*15 100*	*2 451 274*	*3 050 200*	*5.91*	*6.23*
Pepper	*16 115*	*16 409*	*2 618 591*	*3 314 618*	*19.2*	*19.58*
Parrot	*17 728*	*17 728*	*2 856 680*	*3 581 056*	*34.6*	*35*

sur le système *GG-SOM* est inférieur au nombre d'itérations nécessaires pour le système *SOM-NoC statique*, comme présenté dans le tableau 4.6. D'autre part, le temps d'apprentissage du système *GG-SOM* est inférieur à celui du système *SOM-NoC statique*, même si le nombre d'itérations est presque égal. Cela est principalement dû à la taille de la carte *SOM virtuelle* utilisée dans l'architecture *GG-SOM* pendant les premières itérations de la phase d'apprentissage. Nous remarquons également que pour l'image *Lenna*, l'apprentissage du système basé sur la carte auto-organisatrice évolutive *GG-SOM* s'est arrêté avant d'activer tous les neurones physiques sur la

grille. Pour cette image, la structure optimale pour la quantification est composée d'une grille de 8×9 *neurones*. Dans ce cas de figure, l'erreur de quantification moyenne est inférieure au seuil de précision prédéfini $(P_s = 9.45)$.

Les résultats de *reconstruction d'images* obtenus avec les deux systèmes sont présentés à la figure 4.13, pour les quatre images traitées. Pour chaque image, nous pouvons voir la *palette* ainsi que *l'image reconstruite* avec chaque système. D'autre part, le *temps d'apprentissage* T_L et *l'erreur de quantification moyenne* EQM_g sont également présentés.

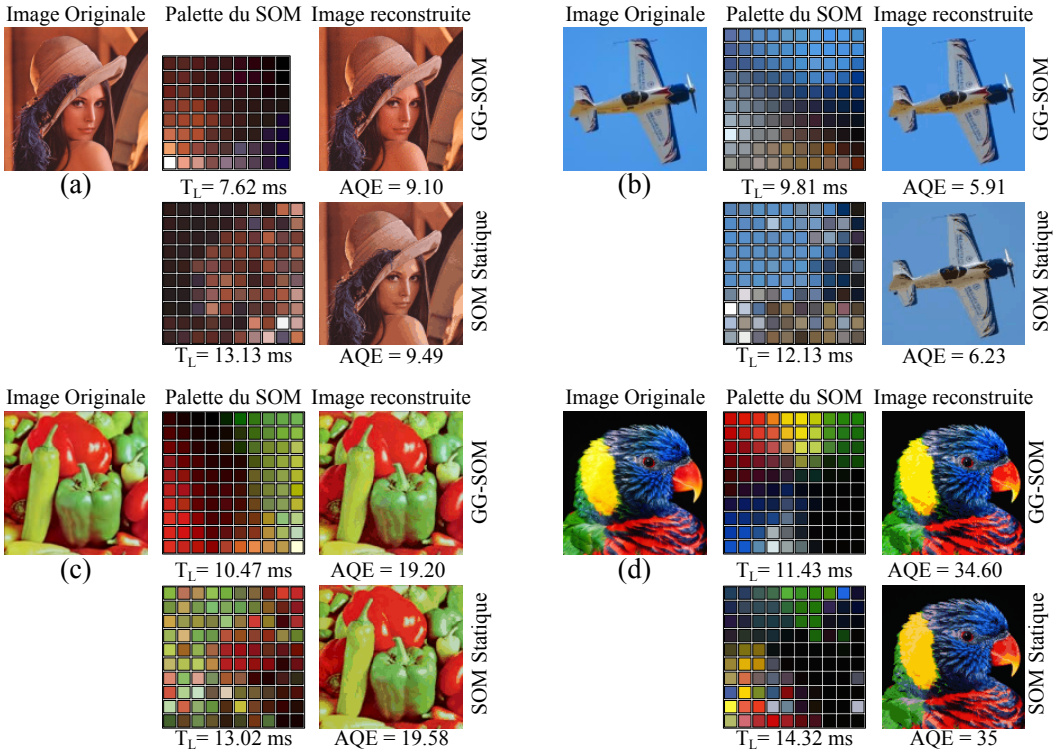


FIGURE *4.13 – Comparaison entre les systèmes de compression d'image utilisant l'approche SOM-NoC statique et Growing Grid SOM respectivement. Résultats de compression des images RGB : (a) Lenna, (b) Airplane, (c) Pepper et (d) Parrot.*

En observant la figure 4.13, on remarque que la *palette* générée par le système *GG-SOM* est mieux organisée et structurée en termes de variation de couleurs que celle générée par le système *SOM-NoC* statique. Chaque couleur est placée avec ces nuances dans la même zone de la palette de couleurs. Cette organisation est le résultat de l'opération d'incrémentation où tous les nouveaux neurones ajoutés à la carte *SOM* se situent, en termes des poids, entre les neurones existants (voir l'équation 4.3). De cette façon, les données d'entrée, étant dans ce cas de figure les pixels ayant des valeurs très proches (des couleurs similaires), seront présentées par les neurones topologiquement proches dans la carte virtuelle finale. D'autre part, cette organisation garantit une convergence rapide (en fonction du nombre d'itérations d'apprentissage) à la précision souhaitée en réduisant les adaptations inapproprié des voisins représentant des classes différentes de celle du neurone gagnant. Par conséquent, la précision obtenue avec un *GG-SOM* est meilleure que celle obtenue avec un *SOM-NoC statique*.

4.4 Conclusion

Dans ce chapitre, nous avons présenté deux nouvelles architectures matérielles de la carte *SOM*, basée sur l'architecture *SOM-NoC* présentée dans le chapitre précédent. La première architecture présentée dans ce chapitre, l'approche *SWAP-SOM*, permet de traiter plusieurs cartes *SOM* ayant des paramètres et structures différents d'une part, et d'offrir une meilleure utilisation et exploitation des neurones de l'architecture d'autre part. L'architecture *SWAP-SOM* hérite de toutes les propriétés de l'architecture *SOM-NoC*, à savoir les propriétés d'extensibilité et d'adaptabilité d'une part, et permet également de limiter l'inconvénient principal de l'architecture *SOM-NoC* étant l'inoccupation importante des neurones lors d'une itération notamment à cause de l'opération de recherche du neurone gagnant distribuée et gourmande en temps d'autre part. L'architecture *SWAP-SOM* permet également d'introduire un degré d'adaptabilité supplémentaire, notamment du point de vue applicatif, puisque les principaux paramètres d'une application donnée ne doivent pas nécessairement être définis à la conception mais peuvent être explorés via ces multiples cartes *SOM* opérant en même temps dans l'architecture SWAP-SOM.

D'un autre coté, nous avons également proposé une première architecture matérielle de la carte *SOM* présentée dans la littérature. Toutes les architectures de la carte *SOM* dites évolutives sont des architectures logicielles puisqu'un degré de flexibilité important est nécessaire pour leur mise en œuvre. L'architecture matérielle de la carte *SOM* évolutive présentée dans ce chapitre et nommée *GG-SOM* offre tous les avantages des architectures précédentes en y ajoutant la possibilité d'évolution au cours du fonctionnement via un mécanisme d'adresses physique et virtuelle couplé à l'approche *SOM-NoC* adaptable présentée dans le chapitre précédent. Les deux architectures, *SWAP-SOM* et *GG-SOM*, ont été validées sur un exemple de système de compression d'images, et comparées avec l'architecture *SOM-NoC* statique.

Les résultats de validation présentés dans ce chapitre ont montré que les deux architectures matérielles dynamiques et évolutives présentent une amélioration significative de l'architecture *SOM-NoC* d'une part, et des cartes auto-organisatrices matérielles de manière générale d'autre part, puisqu'elles introduisent un degré d'adaptabilité aux architectures matérielles de la carte *SOM* jamais présenté dans la littérature. Ces architectures offrent également une avancée significative au niveau applicatif, car elles permettent d'offrir un rapport qualité/performances souhaité de manière dynamique à la volée notamment en fonction des besoins applicatifs du système basé sur ces architectures.

Conclusion générale et perspectives

De part leurs propriétés permettant de modéliser des systèmes complexes en se basant uniquement sur l'observation de leurs comportements en présence des stimuli à leurs entrées, les réseaux de neurones artificiels ont réussi à trouver leur place dans une large gamme d'applications. Plusieurs modèles neuronaux ont été proposés dans la littérature depuis leur première utilisation, chaque modèle étant adapté à une gamme spécifique d'applications. De plus, si l'application finale des réseaux de neurones est destinée à un support matériel embarqué, les performances exprimées en termes de ressources, de consommation d'énergie et de la capacité à respecter les contraintes temps réel doivent également être prises en considération. Pour atteindre ces performances visées, des implémentations matérielles des réseaux de neurones sont souvent proposées comme solution.

Dans la littérature, on trouve plusieurs travaux visant l'implémentation matérielle des réseaux de neurones artificiels. Bien qu'elles permettent de s'auto-organiser pour répondre aux besoins spécifiques d'une application donnée dans les meilleurs délais et avec une consommation d'énergie réduite, les architectures matérielles des réseaux de neurones artificiels souffrent encore d'un manque de flexibilité leur permettant de satisfaire une vaste variété d'applications et de faire face à des changements éventuels de l'environnement dans lequel le système évolue. En effet, les travaux existants cherchent à améliorer l'efficacité d'un modèle neuronal dédié à une fonctionnalité spécifique, sans tenir compte des évolutions fonctionnelles éventuelles du système. De ce fait, pour pouvoir répondre aux nouveaux besoins, le système doit être repensé à nouveau en passant par toutes les phases de conception.

Afin d'ajouter un certain niveau de flexibilité aux architectures matérielles des réseaux de neurones artificiels, des nouvelles stratégies de conception doivent être utilisées afin de proposer des solutions architecturales adaptables et flexibles. De cette façon, les systèmes intégrant des réseaux de neurones artificiels pourraient faire face à des éventuels changements de besoin d'une façon dynamique et au cours du fonctionnement (en ligne ou *online*), sans passer à nouveau par une phase de conception couteuse en termes d'efforts, de temps et de coût. C'est dans ce contexte que se situent les travaux de recherche présentés dans cette thèse, se focalisant en particulier sur la version non supervisée des réseaux de *Kohonen* ou des cartes auto-organisatrices *SOM*.

L'analyse des différentes approches d'implémentation de la carte *SOM* proposées dans la littérature nous a permis d'identifier les différentes causes limitant la flexibilité des architectures matérielles de la carte *SOM*. L'une des principales causes du manque de flexibilité est la technique de communication « *point à point P2P* » souvent utilisée pour les échanges inter-neuronaux. Une deuxième cause consiste en l'utilisation d'un certain nombre de modules globaux, connectés à

tous les neurones de l'architecture, provoquant ainsi une dépendance architecturale forte entre les modules du système et limitant également la possibilité d'étendre de l'architecture en question.

Dans les travaux de recherche menés dans cette thèse, nous avons choisi une technique de communication robuste, extensible et reconfigurable permettant de gérer les changements du comportement global de l'architecture matérielle de la carte *SOM* d'une manière plus aisée et adaptative. De plus, nous avons également fait le choix d'une architecture totalement distribuée à base des modules reconfigurables permettant ainsi de modifier à la volée, de manière dynamique, les paramètres des différentes opérations de l'algorithme de *Kohonen*. Enfin, pour certaines opérations nous avons opté à l'utilisation d'un traitement sériel permettant d'une part, d'assouplir la dépendance architecturale de la structure de la carte *SOM*, et d'autre part, d'améliorer la fréquence globale de fonctionnement et les performances du système.

Dans ce cadre, nous avons proposé une architecture matérielle totalement distribuée de type *MIMD* de la carte *SOM*. L'architecture appelée « *Self-Organizing Map Based on Network-on-Chip* (*SOM-NoC*) » est composée de deux couches, une pour le traitement et une autre pour la communication. La couche de traitement est composée d'un ensemble de neurones et de comparateurs locaux reconfigurables distribués suivant une topologie 2D maillée, sur lesquels les opérations de traitement sont effectuées d'une maniéré sérielle en pipeline. D'autre part, toutes les opérations de communication sont effectuées à base d'un réseau sur puce *NoC* implanté sur une couche séparée, composée d'un ensemble de routeurs dont chacun est associé à un neurone. A base de cette approche, nous avons proposé une architecture matérielle adaptable et extensible de la carte *SOM*. Cette architecture est capable de s'adapter aux besoins de différentes applications par une simple reconfiguration effectuée avant la phase d'apprentissage de la carte *SOM*. Cette reconfiguration permet de modifier à la volée (online) la structure de la carte en termes de nombre de neurones, dimension des vecteurs de poids et la topologie des connexions inter-neuronales. En plus de la flexibilité, les performances temporelles obtenues avec l'architecture proposée sont comparables à celles des architectures matérielles classiques de la carte *SOM* proposées dans la littérature. En termes de *MCPS* et *MCUPS* nous avons obtenu des performances de *18 597 MCUPS* au cours la phase d'apprentissage et *30 913 MCPS* pendant la phase de rappel avec une fréquence de fonctionnement de *250 MHz* sur un support FPGA de type VC707 Virtex-7 de la famille Xilinx. L'architecture proposée a également été validée sur l'exemple d'un système de compression d'image, où l'intérêt de l'adaptabilité de la carte *SOM* aux caractéristiques des images à l'entrée a été démontré.

D'autre part, une architecture matérielle auto-adaptable à base de l'approche *SOM-NoC* a également été proposée dans cette thèse. Cette architecture permet d'exécuter plusieurs cartes *SOM* simultanément sur la même architecture physique. Chaque carte *SOM* exécutée sur l'architecture a ses propres paramètres. Par conséquent, l'architecture appelée « *SWitch APplication Self-Organizing Map* (*SWAP-SOM*) » assure une commutation structurelle, avec une reconfiguration dynamique effectuée d'une façon autonome à la réception d'un échantillon d'entrée indexé par l'identifiant de sa propre application. Cette approche a permis d'améliorer les performances de

l'architecture matérielle de la carte *SOM-NoC* en exploitant le temps d'inoccupation des neurones de traitement via une exécution en pipeline de plusieurs cartes *SOM* propres à différentes applications.

Enfin, une architecture matérielle innovante d'une carte *SOM* à structure croissante au cours de l'apprentissage a été proposée. Cette architecture originale appelée « *Growing Grid Self-Organizing Map* (*GG-SOM*) » permet de surmonter le problème de détermination de la structure optimale de la carte *SOM* à utiliser pour une application donnée. Avec l'architecture *GG-SOM* la carte *SOM* organise sa structure en fonction des échantillons envoyés à l'entrée au cours de l'apprentissage, afin d'aboutir à une structure optimale respectant les critères d'arrêt établis soit par l'utilisateur soit pas les limites physiques, qui sera ensuite utilisée au cours de la phase de rappel.

En conclusion, les travaux menés au cours cette thèse sont donc originaux en plusieurs points. Tout d'abord, parce qu'ils s'insèrent dans une problématique d'actualité traitant des réseaux de neurones à la fois performants et de très grande taille. Dans ce cadre une approche permettant de surmonter les limites de flexibilité et d'extensibilité des implémentations matérielles des réseaux de Kohonen neurones a été proposée. Le deuxième point original des travaux de recherche menés dans cette thèse est la mise en œuvre des architectures adaptables et auto-adaptable de la carte *SOM* permettant d'effectuer plusieurs applications caractérisées par des paramètres différents et de commuter entre elles de manière dynamique à la volée sans repasser par les phases de conception. Enfin, le dernier point original est une première approche matérielle permettant l'implémentation d'un réseau de Kohonen évolutif.

La poursuite des travaux de recherche menés dans le cadre de cette thèse peut être répartie en deux volets : architectural et applicatif. Dans le volet architectural, les travaux de recherche à mener pour améliorer les performances et l'efficacité des architectures proposées, devraient se focaliser sur l'aspect communication en proposant un modèle de réseau sur puce plus adapté aux opérations de communication des cartes *SOM*. Avec cette nouvelle approche de communication adaptée à la carte *SOM*, le temps de communication et la consommation d'énergie pourraient être considérablement réduits. De plus, ce réseau adapté serait à l'origine des meilleures performances globales en phases d'apprentissage et de rappel. D'autre part, dans ce volet architectural des travaux à mener, l'approche « *GG-SOM* », étant une première approche d'architecture matérielle d'un réseau de Kohonen évolutif ou à structure variable dans le temps, pourrait être à l'origine d'autres architectures matérielles des réseaux de Kohonen évolutifs notamment du modèle neuronal « *GNG* », où un nombre minimal de neurones est rajouté à la structure initiale lors de la phase de croissance. De plus, la procédure d'abattage des neurones inactifs dans l'architecture *GG-SOM* pourrait être rajoutée, permettant d'éliminer les neurones peu sélectionnés lors de la phase d'apprentissage. D'un autre côté, dans le volet applicatif des travaux de recherche à mener, les architectures proposées pourraient être utilisées dans la suite des travaux de recherche menés au sein du laboratoire TIM portant sur l'implémentation des systèmes embarqués de détection d'hypovigilance chez les conducteurs à base de la classification des signaux physiologiques « *Elec-*

troEncéphaloGramme (*EEG*) ». Grâce à leur propriété d'adaptabilité, les architectures proposées permettraient de répondre aux besoins de ces systèmes caractérisés par les données hétérogènes spécifiques à l'utilisateur, nécessitant ainsi une phase de réajustement ou de reconfiguration.

Glossaire

ABP *Alternate Bit Protocol*

ADALINE *ADAptative LINear Element*

CHL *Competitive Hebbian Learning*

CPU *Central Processing Unit*

CUDA *Compute Unified Device Architecture*

DCC *Distance Calculation Circuit*

E-SOM *Evolve Self-Organizing Map*

EEG *ElectroEncéphaloGramme*

FPGA *Field-Programmable Gate Array*

Fuzzy ART *Fuzzy Adaptive Resonance Theory*

G-SOM *Growing Self-Organizing Map*

GCM *Global Configuration Management*

GCM *Growing Grid Management*

GCS *Growing Cell Structures*

GG-SOM *Growing Grid Self-Organizing Map*

GH-NN *Growing Herarchical Neural Network*

GH-SOM *Growing Hierarchical Self-organizing Maps*

GNG *Growing Neural Gas*

GN *Neural Gas*

GPU *Graphics Processing Unit*

GSRM *Growing Self-Reconstruction Maps*

GWR *Grow When Required*

I-SOM *Incremental Self-Organizing Map*

IGG-SOM *Incremental Grid Growing Self-Organizing Map*

IOMC *Input/Output Management Circuit*

IP *Intellectual Property*

LCM *Local Configuration Management*

LI *Linear Initialization*

LUT *Look-Up Table*

LVQ *Learning Vector Quantization*

LWS *Local Winner Search*

MCPS *Million Connection Per Second*

MCUPS *Million Connection Updates Per Second*

MIG-SOM *Multilevel Interior Growing Self-Organizing Maps*

MIMD *Multiple Instruction Multiple Data*

MLP *Multi-Layers Perceptron*

MP-SoC *Multi-Processor System-on-Chip*

MSE *Mean Squared Error*

NI *Network Interface*

NoC *Network-on-Chip*

OSI *Open Systems Interconnection*

Open-GL *Open Graphics Library*

P2P *Point-to-Point*

PCI *Principal Components Initialization*

PE *Processing Element*

PL-G-SOM *ParameterLess Growing Self-Organizing Map*

PMC *Perceptron Multi Couches*

PSNR *Peak Signal to Noise Ratio*

RI *Random Initialization*

RWTA *Rough-Winner-Take-All*

SAF *Store-And-Forward*

SIMD *Single Instruction Multiple Data*

SOM-NoC *Self-Organizing Map Based on Network-on-Chip*

SOM *Self-Organising Map*

SWAP-SOM *SWitch APplication Self-Organizing Map*

SoC *System-on-Chip*

TRN *Topology Representing Networks*

TVS-NN *Time-Varying Structure Neural Network*

TVS-SOM *Time-Varying Structure Self-Organizing Map*

USG *Update Signal Generator*

VCT *Virtual-Cut-Trough*

VEP *Vector Element Processor*

VHDL *VHSIC Hardware Description Language*

WS *Wormhole Switching*

Liste des Publications

Revue internationale à comité de lecture (publié) :

1) **M. ABADI**, S. Jovanović, K. Ben Khalifa, S. Weber et M.H. Bédoui « A scalable and adaptable hardware noc-based self organizing map », Journal of Microprocessors and Microsystems, Vol. 57, No. 1, pp. 1-14, 2018.

Revues internationales à comité de lecture (en cours de soumission) :

1) **M. ABADI**, S. Jovanović, K. Ben Khalifa, S. Weber et M.H. Bédoui « A Hardware Growing Grid - a Self Organizing Network-on-Chip based Neural Network », IEEE Transactions on Neural Networks.
2) K. Ben Khalifa, **M. ABADI**, A. G. Blaiech et M.H. Bédoui « A Generic Diagonal Hardware Architecture of Low-Latency Self-Organizing Maps Journal of Real-Time Image Processing », Journal of Systems Architecture.

Conférences internationales avec actes et comité de lecture :

1) **M. ABADI**, S. Jovanović, K. Ben Khalifa, S. Weber et M.H. Bédoui « A hardware configurable self-organizing map for real-time color quantization », IEEE International Conference on Electronics, Circuits and Systems (ICECS), du 11 au 14 Dec 2016 à Monte Carlo, Monaco, pp. 336–339, 2016
2) **M. ABADI**, S. Jovanović, K. Ben Khalifa, S. Weber et M.H. Bédoui « A Scalable Flexible SOM NoC-based Hardware Architecture », 11th International Workshop WSOM 2016, du 6 au 8 Jan 2016 à Houston, Texas, USA, Advances in Self-Organizing Maps and Learning Vector Quantization, pp. 165-175, 2016.

Conférences nationales avec actes et comité de lecture

1) **M. ABADI**, S. Jovanović, K. Ben Khalifa, S. Weber et M.H. Bédoui « Architecture Adaptable et Extensible du Self-OrganizingMap Adaptée à la Commutation d'Applications », Conférence d'informatique en Parallélisme, Architecture et Système (ComPAS), 27 au 30 juin 2017, au campus SophiaTech de Sophia Antipolis, France, 2017.
2) **M. ABADI**, S. Jovanović, K. Ben Khalifa, S. Weber et M.H. Bédoui « A Hardware Adaptable Self-Organizing Map for Real-Time Color Quantization », 12ème Colloque du GDR SoC/SiP,

du 14 au 16 juin 2017 à Bordeaux, France, 2017.

3) **M. ABADI**, S. Jovanović, K. Ben Khalifa, S. Weber et M.H. Bédoui « Architecture du Self-Organizing Map Auto-Adaptative Pour une Application Temps Réel de Quantification de Couleurs », huitième WORKSHOP "Applications Medicales de l'Informatique : Nouvelles Approches (AMINA 2016), du 17 au 19 Nov 2016 à Monastir, Tunisie, 2016.

4) **M. ABADI**, S. Jovanović, K. Ben Khalifa, S. Weber et M.H. Bédoui « A Hardware Design of a Scalable Systolic Self-Organizing Map Using Network on Chip », 11ème Colloque du GDR SoC-SiP, du 8 au 10 Juin 2016 à Nantes, France, 2016.

Bibliographie

[1] I. Manolakos and E. Logaras. High throughput systolic som ip core for fpgas. In *2007 IEEE International Conference on Acoustics, Speech and Signal Processing - ICASSP '07*, volume 2, pages II–61–II–64, April 2007. xiii, 29, 30, 103

[2] M. Porrmann, U. Witkowski, and U. Rückert. *Implementation of Self-Organizing Feature Maps in Reconfigurable Hardware*, pages 247–269. Springer US, Boston, MA, 2006. xiii, 47, 48, 103

[3] J. B. Lagrange. L'intégration d'instruments informatiques dans l'enseignement : une approche par les techniques. *Educational studies in mathematics*, 43(1) :1–30, 2000. 1

[4] M. Duribreux-Cocquebert. *MODESTI : vers une méthodologie interactive de développement de Systèmes à Base de Connaissances*. PhD thesis, Valenciennes, 1995. 1

[5] J. BASQUE and B. PUDELKO. *La modélisation des connaissances à l'aide d'un outil informatisé à des fins de transfert des expertises*. Centre de Recherche LICEF, Université Monréal, Nov 2004. 2

[6] S. H. Liao. Expert system methodologies and applications—a decade review from 1995 to 2004. *Expert systems with applications*, 28(1) :93–103, 2005. 2

[7] D. Kriesel. *A Brief Introduction to Neural Networks*. available at http ://www.dkriesel.com, 2007. 2, 9, 10

[8] C. Touzet. *LES RESEAUX DE NEURONES ARTIFICIELS : INTRODUCTION AU CONNEXIONNISME*. (EPFL) Ecole Polytechnique Fédérale de Lausanne, 1992. 2, 15

[9] J. Liu and D. Liang. A survey of fpga-based hardware implementation of anns. In *2005 International Conference on Neural Networks and Brain*, volume 2, pages 915–918, Oct 2005. 2

[10] M. Forssell. Hardware implementation of artificial neural networks, 2014. 2

[11] KV Ramanaiah and S. Sridhar. Hardware implementation of artificial neural networks. *i-Manager's Journal on Embedded Systems*, 3(4) :31, 2014. 3

[12] S. Jovanovic. *Architecture reconfigurable de système embarqué auto-organisé*. Ecole doctorale informatique, automatique, Électronique - Électrotechnique, mathématiques, université Henri Poincaré – Nancy 1, Nov 2009. Thèse de doctorat dirigée par Weber, Serge Instrumentation et microélectronique Nancy 1 2009. 4, 67, 68, 70, 78

[13] A. Jauffret. *From self-evaluation to emotions : neuromimetic and bayesian approaches for the learning of complex behavior involving multimodal informations*. Theses, Université Paris Sud - Paris XI, July 2014. 8, 17

[14] A. Arleo and W. Gerstner. Spatial cognition and neuro-mimetic navigation : a model of hippocampal place cell activity. *Biological Cybernetics*, 83(3) :287–299, 2000. 8

[15] K. Caluwaerts, M. Staffa, S. N'Guyen, C. Grand, L. Dollé, A. Favre-Félix, B. Girard, and M. Khamassi. A biologically inspired meta-control navigation system for the psikharpax rat robot. *Bioinspiration and Biomimetics*, 7(2) :025009, May 2012. 8

[16] M. Ranciaro, G. Nunes Nogueira Neto, C. Roberto Fernandes, J. Carlos da Cunha, and P. Nohama. Mimetic motion control for a lower-extremity active orthosis for hemiplegic people. *IEEE Latin America Transactions*, 15(2) :225–231, Feb 2017. 8

[17] M. Seo, H. Kim, and Y. Choi. Human mimetic forearm mechanism towards bionic arm. In *2017 International Conference on Rehabilitation Robotics (ICORR)*, pages 1171–1176, July 2017. 8

[18] B. Victorri. Chapitre 7. le connexionnisme. *Traité de neuropsychologie clinique. De Boeck Supérieur*, 1 :53–64, April 2008. 8, 9

[19] A. Roy. Connectionism, controllers, and a brain theory. *IEEE Transactions on Systems, Man, and Cybernetics - Part A : Systems and Humans*, 38(6) :1434–1441, Nov 2008. 8

[20] S. W. K. Chan and J. Franklin. Symbolic connectionism in natural language disambiguation. *IEEE Transactions on Neural Networks*, 9(5) :739–755, Sep 1998. 8

[21] B. Krose and P. Van-Der-Smagt. *An Introduction to Neural Networks*. Eighth edition, 1996. 8, 9

[22] Larousse. Dictinnaire de français. *http ://www.larousse.fr/dictionnaires/francais*, 2001 :site web, January 2018. 9, 13, 15

[23] N. Tsopze. *Treillis de Galois et réseaux de neurones : une approche constructive d'architecture des réseaux de neurones*. PhD thesis, Université de Yaoundé I, 2010. 9, 13

[24] W.S. McCulloch and W. Pitts. A logical calculus of the ideas immanent in nervous activity. *The bulletin of mathematical biophysics*, 5(4) :115–133, 1943. 10

[25] D.O. Hebb. *The Organization of Behavior*. A Neuropsychological Theory, 1949. Lawrance Erlbaum Associates. 10, 19

[26] F. Rosenblatt. The perceptron : A probabilistic model for information storage and organization in the brain. *Psychological Review*, 2 :386–408, 1958. 11

[27] B. Widrow and M.E. Hoff. Adaptive switching circuits. *IRE WESCON Convention Record*, 2 :96–104, 1960. Reprinted in *Neurocomputing* MIT Press, 1988 . 11, 19

[28] M. Minsky and S. Papert. *Perceptrons : An Introduction to Computational Geometry*. MIT Press, Cambridge, MA, USA, 1969. 11

[29] J. J. Hopfield. Neural networks and physical systems with emergent collective computational abilities. *Proc. of the National Academy of Science*, 79 :25542558, 1982. 11

[30] T. Kohonen. Self-organized formation of topologically correct feature maps. *Biological Cybernetics*, 43(1) :59–69, 1982. 11

[31] K. Fukushima, S. Miyake, and T. Ito. Neocognitron : A neural network model for a mechanism of visual pattern recognition. *IEEE Transactions on Systems, Man, and Cybernetics*, SMC-13(5) :826–834, Sept 1983. 11

[32] D.B. Parker. *Learning Logic : Casting the Cortex of the Human Brain in Silicon*, volume 47 of *Technical report : Center for Computational Research in Economics and Management Science*. Center for Computational Research in Economics and Management Science, 1985. 11, 15

[33] D.E. Rumelhart, G.E. Hinton, and R.J. Williams. *Learning Internal Representations by Error Propagation*. ICS report. Institute for Cognitive Science, University of California, San Diego, 1986. 11

[34] Yann Lecun. *A learning scheme for asymmetric threshold networks*, pages 599–604. Proceedings of Cognitiva 85, Paris, France, 1985. 11

[35] E. Daucé. *Dynamic adaptation and learning in random recurrent neural networks*. Theses, ISAE - Institut Supérieur de l'Aéronautique et de l'Espace, January 2000. 12, 14

[36] J.P. Rennard. *Réseaux neuronaux : Une introduction*. Vuibert, 2006. 12

[37] R. Rojas. *Neural Networks : A Systematic Introduction*. Number 1 in Artificial Intelligence (incl. Robotics). Springer-Verlag Berlin Heidelberg, New-York, 1996. 14

[38] Y. Bennani. *Apprentissage connexionniste*. informatique et systeme d'information. Hermes - Lavoisier, Jan 2006. 14

[39] W. Duch and N. Jankowski. Survey of neural transfer functions. *Neural Computing Surveys*, 2 :163–213, 1999. 15

[40] J.L. Amat, G. Yahiaoui, and H. Morice. *Techniques avancées pour le traitement de l'information : réseaux de neurones, logique floue, algorithmes génétiques*. Cépaduès, 2002. 16

[41] M. Parizeau. *RESEAUX DE NEURONES*. Université LAVAL, 2004. 16, 22

[42] I. PAVLOV. *Conditioned reflexes : An investigation of the physiological activity of the cerebral cortex*. OXFORD UNIVERSITY PRESS : HUMPHREY MILFORD, 1927. 17

[43] F. Elie. *Conception et réalisation d'un système utilisant des réseaux de neurones pour l'identification et la caractérisation, à bord de satellites, des signaux transitoire de type sifflement*. PhD thesis, Université Orléans, France, 1997. 17

[44] K. B. Khalifa. *Traitement matériel et logiciel des signaux physiologiques : application à l'étude de la vigilance*. PhD thesis, Université de Monastir, 2006. 17

[45] S. Haykin. *Neural Networks : A Comprehensive Foundation*. Prentice Hall PTR, Upper Saddle River, NJ, USA, 2nd edition, 1998. 17, 19

[46] J.B. MacQueen. Some applications of monotone operators in markov processes. *Annals of Mathematical Statistics*, 36 :1421–1425, 1965. 19

[47] T. Kohonen. *Associative Memory : A System-Theoretical Approach*, volume 17. Springer-Verlag Berlin Heidelberg, communication and cybernetics edition, 1978. 19, 38

[48] T. Kohonen. Analysis of a simple self-organizing process. *Biological Cybernetics*, 44(2) :135–140, 1982. 20, 27, 38

[49] S. Sakavicius, D. Plonis, and A. Serackis. Single sound source localization using multilayer perceptron. In *2017 Open Conference of Electrical, Electronic and Information Sciences (eStream)*, pages 1–4, April 2017. 20

[50] S. Hui and S. H. Zak. Robust stability analysis of adaptation algorithms for single perceptron. *IEEE Transactions on Neural Networks*, 2(2) :325–328, Mar 1991. 20

[51] M.W Gardner and S.R Dorling. Artificial neural networks (the multilayer perceptron)—a review of applications in the atmospheric sciences. *Atmospheric Environment*, 32(14) :2627 – 2636, 1998. 21

[52] J. Bayer, S.S. Bukhari, and A. Dengel. Floor plan generation and auto completion based on recurrent neural networks. In *2017 14th IAPR International Conference on Document Analysis and Recognition (ICDAR)*, volume 02, pages 49–50, Nov 2017. 22

[53] A. R. Da Silva and L. F. Wanderley Goes. Hearthbot : An autonomous agent based on fuzzy art adaptive neural networks for the digital collectible card game hearthstone. *IEEE Transactions on Computational Intelligence and AI in Games*, PP(99) :1–1, 2017. 22

[54] T. Kohonen. *Self-Organization and Associative Memory*, volume 8 of *Springer Series in Information Sciences*. Springer Berlin Heidelberg, 2 edition, 1988. 23, 38

[55] T. Kohonen. *Self-Organizing Maps, 3rd edition*. Springer Series in Information Sciences. Springer Berlin Heidelberg, 2001. 23, 24, 27, 35, 38, 45, 61, 81, 82, 83

[56] T. M. Martinetz, S. G. Berkovich, and K. J. Schulten. 'neural-gas' network for vector quantization and its application to time-series prediction. *IEEE Transactions on Neural Networks*, 4(4) :558–569, Jul 1993. 23, 125

[57] R.H. White. Competitive hebbian learning : Algorithm and demonstrations. *Neural Networks*, 5(2) :261 – 275, 1992. 23

[58] T. Kohonen. Improved versions of learning vector quantization. In *1990 IJCNN International Joint Conference on Neural Networks*, pages 545–550 vol.1, June 1990. 23

[59] K. Garcia and C.H.Q. Forster. Supervised growing neural gas. In Hujun Yin, José A. F. Costa, and Guilherme Barreto, editors, *Intelligent Data Engineering and Automated Learning - IDEAL 2012*, pages 502–507, Berlin, Heidelberg, 2012. Springer Berlin Heidelberg. 23

[60] T. Kohonen. *Learning Vector Quantisation and the Self Organising Map*, pages 235–242. Springer London, London, 1992. 23, 24, 27, 35, 39

[61] T. Kohonen. The neural phonetic typewriter. *Computer*, 21(3) :11–22, March 1988. 24

[62] D. Heinke and F. H. Hamker. Comparing neural networks : a benchmark on growing neural gas, growing cell structures, and fuzzy artmap. *IEEE Transactions on Neural Networks*, 9(6) :1279–1291, Nov 1998. 24

[63] Y. Prudent and A. Ennaji. An incremental growing neural gas learns topologies. In *Proceedings. 2005 IEEE International Joint Conference on Neural Networks, 2005.*, volume 2, pages 1211–1216 vol. 2, July 2005. 24

[64] Bernd Fritzke. Growing cell structures—a self-organizing network for unsupervised and supervised learning. *Neural Networks*, 7(9) :1441 – 1460, 1994. 24, 125

[65] C. Von Der Malsburg. Self-organization of orientation sensitive cells in the striate cortex. *Kybernetik*, 14(2) :85–100, Jun 1973. 26, 27

[66] D. J. Willshaw and C. Von Der Malsburg. How patterned neural connections can be set up by self-organization. In *the Royal Society of London. Series B, Biological Sciences*, volume 194, pages 431–445, Nov 1976. 26

[67] C. Von Der Malsburg. How to label nerve cells so that they can interconnect in an ordered fashion. In *the National Academy of Sciences of the United States of America*, volume 74, pages 5176–5178, 1977. 26

[68] D. J. Willshaw and C. Von Der Malsburg. A marker induction mechanism for the establishment of ordered neural mappings : its application to the retinotectal problem. *Philosophical Transactions of the Royal Society of London*, 287(1021) :203–243, Vov 1979. 26

[69] K. BenAbdeslem. *Approches connexionnistes pour la visualisation et la classification des séquences évolutives : Application aux données issues d'usages d'Internet*. PhD thesis, Université Paris 13, 2003. 28, 35, 57

[70] H. Zhou and T. Kawamura. Linear distortion compensation based on som for digital wireless communications. In *IEEE International Symposium on Communications and Information Technology, 2004. ISCIT 2004.*, volume 2, pages 1155–1159 vol.2, Oct 2004. 29, 30

[71] F. Coleca, A. State, S. Klement, E. Barth, and T. Martinetz. Self-organizing maps for hand and full body tracking. *Neurocomputing*, 147 :174 – 184, 2015. Advances in Self-Organizing Maps Subtitle of the special issue : Selected Papers from the Workshop on Self-Organizing Maps 2012 (WSOM 2012). 30

[72] J. Pena, M. Vanegas, and A. Valencia. Digital hardware architectures of kohonen's self organizing feature maps with exponential neighboring function. In *2006 IEEE International Conference on Reconfigurable Computing and FPGA's (ReConFig 2006)*, pages 1–8, Sept 2006. 30, 31, 46, 47, 82

[73] D. Wijayasekara, O. Linda, and M. Manic. Cave-som : Immersive visual data mining using 3d self-organizing maps. In *The 2011 International Joint Conference on Neural Networks*, pages 2471–2478, July 2011. 31, 39

[74] J. Barhak and A. Fischer. Adaptive reconstruction of freeform objects with 3d som neural network grids. *Computers & Graphics*, 26(5) :745 – 751, 2002. 31

[75] N. Sudha, T. Srikanthan, and B. Mailachalam. A vlsi architecture for 3-d self-organizing map based color quantization and its fpga implementation. *Journal of Systems Architecture*, 48(11) :337 – 352, 2003. 31

[76] H. Hikawa, K. Doumoto, S. Miyoshi, and Y. Maeda. Image compression with hardware self-organizing map. In *The 2010 International Joint Conference on Neural Networks (IJCNN)*, pages 1–8, July 2010. 31

[77] C. Shi, J. Yang, Y. Han, Z. Cao, Q. Qin, L. Liu, N. J. Wu, and Z. Wang. A 1000 fps vision chip based on a dynamically reconfigurable hybrid architecture comprising a pe array processor and self-organizing map neural network. *IEEE Journal of Solid-State Circuits*, 49(9) :2067–2082, Sept 2014. 31

[78] M.C. Su, T.K. Liu, and H.T. Chang. Improving the selforganizing feature map algorithm using an efficient initialization scheme. *Tamkang Journal of Science and Engineering*, 5(1) :35–48, 2002. 31

[79] E. Mohebi and A. Bagirov. Modified self-organising maps with a new topology and initialisation algorithm. *Journal of Experimental & Theoretical Artificial Intelligence*, 27(3) :351–372, 2015. 31, 33

[80] J. M. Barbalho, A. Duarte, D. Neto, J. A. F. Costa, and M. L. A. Netto. Hierarchical som applied to image compression. In *Neural Networks, 2001. Proceedings. IJCNN '01. International Joint Conference on*, volume 1, pages 442–447 vol.1, 2001. 31

[81] R. Dlugosz, M. Kolasa, W. Pedrycz, and M. Szulc. Parallel programmable asynchronous neighborhood mechanism for kohonen som implemented in cmos technology. *IEEE Transactions on Neural Networks*, 22(12) :2091–2104, Dec 2011. 32, 46

[82] A. Onoo, H. Hikawa, S. Miyoshi, and Y. Maeda. On automatic generation of vhdl code for self-organizing map. In *2009 International Joint Conference on Neural Networks*, pages 2366–2373, June 2009. 32

[83] P. Kolinummi, P. Pulkkinen, T. Hamalainen, and J. Saarinen. Parallel implementation of self-organizing map on the partial tree shape neurocomputer. *Neural Processing Letters*, 12(2) :171–182, Oct 2000. 32, 39, 42

[84] M. Kolasa, R. Długosz, W. Pedrycz, and M. Szulc. A programmable triangular neighborhood function for a kohonen self-organizing map implemented on chip. *Neural Networks*, 25 :146 – 160, 2012. 32, 48

[85] A. Cherif. *Réseaux de neurones, SVM et approches locales pour la prévision des séries temporelles*. PhD thesis, Université François Rabelais de Tours, 2013. 32

[86] M.C. Su, T.K. Liu, and H.T. Chang. An efficient initialization scheme for the self-organizing feature map algorithm. In *Neural Networks, 1999. IJCNN '99. International Joint Conference on*, volume 3, pages 1906–1910 vol.3, 1999. 33

[87] H. Haripriya, R. Devisree, D. Pooja, and P. Nedungadi. A comparative performance analysis of self organizing maps on weight initializations using different strategies. In *2015 Fifth International Conference on Advances in Computing and Communications (ICACC)*, pages 434–438, Sept 2015. 33

[88] V. Aggarwal, A.K. Ahlawat, and B.N. Pandey. A weight initialization approach for training self organizing maps for clustering applications. In *2013 3rd IEEE International Advance Computing Conference (IACC)*, pages 1000–1005, Feb 2013. 33

[89] R. Gray. Vector quantization. *IEEE ASSP Magazine*, 1(2) :4–29, April 1984. 39

[90] X. Li and Y. Zhang. Digital image edge detection based on lvq neural network. In *2016 IEEE 11th Conference on Industrial Electronics and Applications (ICIEA)*, pages 1251–1255, June 2016. 39

[91] E. C. Djamal and P. Lodaya. Eeg based emotion monitoring using wavelet and learning vector quantization. In *2017 4th International Conference on Electrical Engineering, Computer Science and Informatics (EECSI)*, pages 1–6, Sept 2017. 39

[92] H. Liu, N. Stoll, S. Junginger, and K. Thurow. Human face orientation recognition for intelligent mobile robot collision avoidance in laboratory environments using feature detection and lvq neural networks. In *2015 IEEE International Conference on Robotics and Biomimetics (ROBIO)*, pages 2003–2007, Dec 2015. 39

[93] L.P. Chen, Y.G. Liu, Z.X. Huang, and Y.T. Shi. An improved som algorithm and its application to color feature extraction. *Neural Computing and Applications*, 24(7) :1759–1770, Jun 2014. 39, 63

[94] T. Kuremoto, T. Otani, M. Obayashi, K. Kobayashi, and S. Mabu. A hand shape instruction recognition and learning system using growing som with asymmetric neighborhood function. *Neurocomputing*, 188 :31 – 41, 2016. Advanced Intelligent Computing Methodologies and Applications. 39, 58, 63, 125, 128

[95] T. Hämäläinen, H. Klapuri, J. Saarinen, and K. Kaski. Mapping of som and lvq algorithms on a tree shape parallel computer system. *Parallel Computing*, 23(3) :271 – 289, 1997. 39, 41, 42, 63

[96] F. C. Moraes, S. C. Botelho, N. D. Filho, and J. F. O. Gaya. Parallel high dimensional self organizing maps using cuda. In *2012 Brazilian Robotics Symposium and Latin American Robotics Symposium*, pages 302–306, Oct 2012. 39, 43

[97] Y. Xiao, R.B. Feng, Z.F. Han, and C.S. Leung. Gpu accelerated self-organizing map for high dimensional data. *Neural Processing Letters*, 41(3) :341–355, Jun 2015. 39, 43, 63

[98] T. Richardson and E. Winer. Extending parallelization of the self-organizing map by combining data and network partitioned methods. *Advances in Engineering Software*, 88 :1 – 7, 2015. 39, 43

[99] A. De, Y. Zhang, and C. Guo. A parallel adaptive segmentation method based on som and gpu with application to mri image processing. *Neurocomputing*, 198 :180 – 189, 2016. Advances in Neural Networks, Intelligent Control and Information Processing. 39, 43

[100] B. Fritzke. Growing grid - a self-organizing network with constant neighborhood range and adaptation strength. *Neural Processing Letters*, 2(5) :9–13, Sep 1995. 39, 40, 130

[101] A. Rauber, D. Merkl, and M. Dittenbach. The growing hierarchical self-organizing map : exploratory analysis of high-dimensional data. *IEEE Transactions on Neural Networks*, 13(6) :1331–1341, Nov 2002. 39, 41, 126

[102] H. Sasamura and T. Saito. A simple learning algorithm for growing self-organizing maps and its application to the skeletonization. In *Proceedings of the International Joint Conference on Neural Networks, 2003.*, volume 1, pages 787–790 vol.1, July 2003. 39, 40

[103] B. Conan-Guez, F. Rossi, and A. El-Golli. Fast algorithm and implementation of dissimilarity self-organizing maps. *Neural Networks*, 19(6) :855 – 863, 2006. Advances in Self Organising Maps - WSOM'05. 39, 41

[104] Y. Wang, A. Peyls, Y. Pan, L. Claesen, and X. Yan. A fast self-organizing map algorithm for handwritten digit recognition. In James J. (Jong Hyuk) Park, Joseph Kee-Yin Ng, Hwa-Young Jeong, and Borgy Waluyo, editors, *Multimedia and Ubiquitous Engineering*, pages 177–183, Dordrecht, 2013. Springer Netherlands. 39, 41

[105] U. Seiffert. Artificial neural networks on massively parallel computer hardware. *Neurocomputing*, 57 :135 – 150, 2004. New Aspects in Neurocomputing : 10th European Symposium on Artificial Neural Networks 2002. 39, 43, 63

[106] J. Lachmair, E. Merényi, M. Porrmann, and U. Rückert. A reconfigurable neuroprocessor for self-organizing feature maps. *Neurocomputing*, 112 :189 – 199, 2013. Advances in artificial neural networks, machine learning, and computational intelligence. 39, 40, 44, 46, 53, 63, 103

[107] A.R. Omondi and J.C. Rajapakse, editors. *FPGA Implementations of Neural Networks*. Number 1 in Circuits and Systems. Springer US, 2006. 39, 43, 44

[108] T. Talaśka, M. Kolasa, R. Długosz, and W. Pedrycz. Analog programmable distance calculation circuit for winner takes all neural network realized in the cmos technology. *IEEE Transactions on Neural Networks and Learning Systems*, 27(3) :661–673, March 2016. 39, 47, 81

[109] C. Popa. Current-mode euclidean distance circuit independent on technological parameters. In *CAS 2005 Proceedings. 2005 International Semiconductor Conference, 2005.*, volume 2, pages 459–462 vol. 2, Oct 2005. 39, 45

[110] W. Kurdthongmee. Utilization of a fast mse calculation approach to improve the image quality and accelerate the operation of a hardware k-som quantizer. *Microprocessors and Microsystems*, 34(6) :174 – 181, 2010. 39, 45, 63

[111] A. Ramirez-Agundis, R. Gadea-Girones, and R. Colom-Palero. A hardware design of a massive-parallel, modular nn-based vector quantizer for real-time video coding. *Journal of Microprocessors and Microsystems*, 32(1) :33 – 44, 2008. 39, 45, 48, 49, 51, 63, 103, 106

[112] M. Porrmann, U. Witkowski, H. Kalte, and U. Ruckert. Implementation of artificial neural networks on a reconfigurable hardware accelerator. In *Proceedings 10th Euromicro Workshop on Parallel, Distributed and Network-based Processing*, pages 243–250, 2002. 39, 45

[113] F. An, X. Zhang, L. Chen, and H. J. Mattausch. A memory-based modular architecture for som and lvq with dynamic configuration. *IEEE Transactions on Multi-Scale Computing Systems*, 2(4) :234–241, Oct 2016. 39, 46, 53, 59, 61, 63

[114] W. Kurdthongmee. A low latency minimum distance searching unit of the som based hardware quantizer. *Microprocessors and Microsystems*, 39(2) :135 – 143, 2015. 39, 46, 61

[115] H. Hikawa and Y. Maeda. Improved learning performance of hardware self-organizing map using a novel neighborhood function. *IEEE Transactions on Neural Networks and Learning Systems*, 26(11) :2861–2873, Nov 2015. 39, 47, 51, 63, 103, 104, 106

[116] H. Hikawa and K. Kaida. Novel fpga implementation of hand sign recognition system with som hebb classifier. *IEEE Transactions on Circuits and Systems for Video Technology*, 25(1) :153–166, Jan 2015. 39, 56, 61, 63, 104

[117] M. Porrmann, U. Witkowski, and U. Ruckert. A massively parallel architecture for self-organizing feature maps. *IEEE Transactions on Neural Networks*, 14(5) :1110–1121, Sept 2003. 39, 51

[118] H. Tamukoh, T. Koga, K. Horio, and T. Yamakawa. Rough-winner-take-all self-organizing neural network for hardware oriented vector quantization algorithm. In *2007 50th Midwest Symposium on Circuits and Systems*, pages 349–352, Aug 2007. 39, 48, 59, 61

[119] H. Tamukoh and M. Sekine. A dynamically reconfigurable platform for self-organizing neural network hardware. In Kok Wai Wong, B. Sumudu U. Mendis, and Abdesselam Bouzerdoum, editors, *Neural Information Processing. Models and Applications*, pages 439–446, Berlin, Heidelberg, 2010. Springer Berlin Heidelberg. 39, 59, 63, 103

[120] H. Tamukoh, T. Aso, K. Horio, and T. Yamakawa. Self-organizing map hardware accelerator system and its application to realtime image enlargement. In *2004 IEEE International Joint Conference on Neural Networks (IEEE Cat. No.04CH37541)*, volume 4, pages 2683–2687 vol.4, July 2004. 39, 53, 103

[121] D.C. Hendry and R. Cambio. Techniques for power reduction in an simd implementation of the vq/som algorithms. *Neurocomputing*, 74(3) :291–300, 2010. 39, 53, 54, 55

[122] P. Arato, Z. A. Mann, Z. D. Mann, and A. Orban. Hardware-software co-design for kohonen's self-organizing map. *Intelligent systems*, 1 :1–12, 2003. 39

[123] P. Thole, H. Speckmann, and W. Rosenstiel. A hardware supported system for kohonen's self organizing map. In *international Conf. on Microelectronics for Neural Networks*, pages 29–34, 1993. 39

[124] Teuvo Kohonen, Jussi Hynninen, Jari Kangas, and Jorma Laaksonen. Som pak : The self-organizing map program package. *Report A31, Helsinki University of Technology, Laboratory of Computer and Information Science*, 1 :1–27, 1996. 39, 40

[125] K. Ben Khalifa, B. Girau, F. Alexandre, and M. H. Bedoui. Parallel fpga implementation of self-organizing maps. In *Proceedings. The 16th International Conference on Microelectronics, 2004. ICM 2004.*, pages 709–712, Dec 2004. 47, 48, 50, 63

[126] S. Rüping, M. Porrmann, and U. Rückert. Som accelerator system. *Neurocomputing*, 21(1) :31 – 50, 1998. 47, 82

[127] Takashi Kuremoto, Takahito Komoto, Kunikazu Kobayashi, and Masanao Obayashi. Parameterless-growing-som and its application to a voice instruction learning system. *Journal of Robotics*, 2010 :1–10, 2010. 48, 125, 128

[128] W. Kurdthongmee. A hardware centric algorithm for the best matching unit searching stage of the som-based quantizer and its fpga implementation. *J. Real-Time Image Processing*, 12(1) :71–80, 2016. 55, 106

[129] W. Kurdthongmee. A novel hardware-oriented kohonen som image compression algorithm and its fpga implementation. *Journal of Systems Architecture*, 54(10) :983 – 994, 2008. 56, 61

[130] R. Wang. Self-organized patterns in the som network. In *2015 11th International Conference on Natural Computation (ICNC)*, pages 123–128, Aug 2015. 57

[131] V. Haghighatdoost and R. Safabakhsh. Automatic multilevel color image thresholding by the growing time adaptive self organizing map. In *2006 2nd International Conference on Information Communication Technologies*, volume 1, pages 1768–1772, 2006. 58, 125

[132] A. F. R. Araujo and O. V. Santana. Self-organizing map with time-varying structure to plan and control artificial locomotion. *IEEE Transactions on Neural Networks and Learning Systems*, 26(8) :1594–1607, Aug 2015. 58, 125

[133] S. Orts-Escolano, J. Garcia-Rodriguez, V. Moreli, M. Cazorla, and J. M. Garcia-Chamizo. 3d colour object reconstruction based on growing neural gas. In *2014 International Joint Conference on Neural Networks (IJCNN)*, pages 1474–1481, July 2014. 58, 125

[134] J. García-Rodríguez, A. Angelopoulou, J.M. García-Chamizo, A. Psarrou, S.O. Escolano, and V.M. Giménez. Autonomous growing neural gas for applications with time constraint : Optimal parameter estimation. *Neural Networks*, 32 :196 – 208, 2012. Selected Papers from IJCNN 2011. 58, 126

[135] M. Kolasa, T. Talaska, and R. Długosz. A novel recursive algorithm used to model hardware programmable neighborhood mechanism of self-organizing neural networks. *Applied*

Mathematics and Computation, 267 :314 – 328, 2015. The Fourth European Seminar on Computing (ESCO 2014). 58, 63

[136] L. Vojacek, J. Dvorsky, K. Slaninoa, and J. Martinovic. Scalable parallel som learning for web user profiles. In *13th International Conference on Intellient Systems Design and Applications*, pages 283–288, Dec 2014. 60

[137] Richard D. Lawrence, George Almási, and Holly E. Rushmeier. A scalable parallel algorithm for self-organizing maps with applications to sparse data mining problems. *Data Mining and Knowledge Discovery*, 3 :171–195, 1999. 60

[138] H. Hikawa. Improved winner-take-all circuit for neural network based on frequency-modulated signals. In *2016 IEEE International Conference on Electronics, Circuits and Systems (ICECS)*, pages 85–88, Dec 2016. 61

[139] J. Mazurkiewicz. Systolic realization of kohonen neural network. In *Artificial Neural Networks : Formal Models and Their Applications – ICANN 2005*, pages 1015–1020, Berlin, Heidelberg, 2005. Springer Berlin Heidelberg. 61, 63, 83

[140] K. Yamamoto, Y. Oba, Z. Rikuhashi, and H. Hikawa. Automatic generation of hardware self-organizing map for fpga implementation. In *2011 International Symposium on Intelligent Signal Processing and Communications Systems (ISPACS)*, pages 1–6, Dec 2011. 63

[141] G. Marwa, B. Mohamed, C. Najoua, and B. M. Hédi. Generic neural architecture for lvq artificial neural networks. In *2017 International Conference on Engineering MIS (ICEMIS)*, pages 1–7, May 2017. 63

[142] J. Hu, Y. Deng, and R. Marculescu. System-level point-to-point communication synthesis using floorplanning information [soc]. In *Proceedings of ASP-DAC/VLSI Design . 7th Asia and South Pacific Design Automation Conference and 15h International Conference on VLSI Design*, pages 573–579, 2002. 67

[143] H.Y. Hsieh, B.W. Chen, and T.C. Wang. An improved methodology for system-level point-to-point communication architecture synthesis in soc design. In *IEEE VLSI-TSA International Symposium on VLSI Design, Automation and Test, (VLSI-TSA-DAT).*, pages 192–195, April 2005. 67

[144] S. Pasricha, N. D. Dutt, E. Bozorgzadeh, and M. Ben-Romdhane. Fabsyn : floorplan-aware bus architecture synthesis. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 14(3) :241–253, March 2006. 68

[145] N. Fern, I. San, C. K. Koc, and K. T. T. Cheng. Hiding hardware trojan communication channels in partially specified soc bus functionality. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, PP(99) :1–1, 2017. 68

[146] A. Shrivastava and S. K. Sharma. Various arbitration algorithm for on-chip(amba) shared bus multi-processor soc. In *IEEE Students Conference on Electrical, Electronics and Computer Science (SCEECS)*, pages 1–7, March 2016. 68

[147] B. Tiwari, R. Chandraker, and N. Goel. Comparative analysis of different lottery bus arbitration techniques for soc communication. In *International Conference on Computational Techniques in Information and Communication Technologies (ICCTICT)*, pages 495–499, March 2016. 69

[148] L. Benini and G. De Micheli. Networks on chips : a new soc paradigm. *Computer*, 35(1) :70–78, Jan 2002. 70

[149] W. J. Dally and B. Towles. Route packets, not wires : on-chip interconnection networks. In *Proceedings of the 38th Design Automation Conference*, pages 684–689, 2001. 70, 76

[150] P. Guerrier and A. Greiner. A generic architecture for on-chip packet-switched interconnections. In *Proceedings Design, Automation and Test in Europe Conference and Exhibition*, pages 250–256, 2000. 70, 78

[151] A. Andriahantenaina and A. Greiner. Micro-network for soc : implementation of a 32-port spin network. In *Design, Automation and Test in Europe Conference and Exhibition*, pages 1128–1129, 2003. 70, 78

[152] F. Karim, A. Nguyen, and S. Dey. An interconnect architecture for networking systems on chips. *IEEE Micro*, 22(5) :36–45, Sep 2002. 70, 76

[153] S. H. Sfar, I. E. Bennour, and R. Tourki. Transaction level modeling of an osi-like layered noc. In *International Conference on Design and Test of Integrated Systems in Nanoscale Technology, DTIS*, pages 404–408, Sept 2006. 73

[154] J. Duato, S. Yalamanchili, and L. Ni. *Interconnection Networks : An Engineering Approach*. Morgan Kaufmann, 1st edition edition, July 2003. 74, 75, 76

[155] C. M. Wu and H. C. Chi. Design of a high-performance switch for circuit-switched on-chip networks. In *IEEE Asian Solid-State Circuits Conference*, pages 481–484, Nov 2005. 74

[156] A. Naik and T. K. Ramesh. Efficient network on chip (noc) using heterogeneous circuit switched routers. In *International Conference on VLSI Systems, Architectures, Technology and Applications (VLSI-SATA)*, pages 1–6, Jan 2016. 74

[157] S. Liu, A. Jantsch, and Z. Lu. Analysis and evaluation of circuit switched noc and packet switched noc. In *Euromicro Conference on Digital System Design*, pages 21–28, Sept 2013. 74

[158] S. Kumar, A. Jantsch, J. P. Soininen, M. Forsell, M. Millberg, J. Oberg, K. Tiensyrja, and A. Hemani. A network on chip architecture and design methodology. In *Proceedings IEEE Computer Society Annual Symposium on VLSI. New Paradigms for VLSI Systems Design. ISVLSI*, pages 105–112, 2002. 75

[159] H. Kariniemi and J. Nurmi. *Arbitration and Routing Schemes for on-Chip Packet Networks*, pages 253–282. Springer US, Boston, MA, 2005. 75, 76

[160] W. Dally and B. Towles. *Principles and Practices of Interconnection Networks*. Morgan Kaufmann, 1st edition edition, Dec 2003. 75

[161] J. Hu and R. Marculescu. Dyad - smart routing for networks-on-chip. In *Proceedings. 41st Design Automation Conference*, pages 260–263, July 2004. 75

[162] J. Kim, D. Park, T. Theocharides, N. Vijaykrishnan, and C. R. Das. A low latency router supporting adaptivity for on-chip interconnects. In *Proceedings. 42nd Design Automation Conference*, pages 559–564, June 2005. 75

[163] F. Karim, A. Nguyen, S. Dey, and R. Rao. On-chip communication architecture for oc-768 network processors. In *Proceedings of the 38th Design Automation Conference*, pages 678–683, June 2001. 76

[164] P. P. Pande, C. Grecu, A. Ivanov, and R. Saleh. Design of a switch for network on chip applications. In *Circuits and Systems, ISCAS '03. Proceedings of the 2003 International Symposium on*, volume 5, pages 217–220, May 2003. 78

[165] A. Allen N. Lightowler, C. Spracklen. A modular approach to implementation of the self-organising map. In *Proceedings of WSOM'97*, volume 3, pages 130–135, 1997. 82

[166] P. A. Mei, C. de Carvalho Carneiro, M. C. Kuroda, S. J. Fraser, L. L. Min, and F. Reis. Self-organizing maps as a tool for segmentation of magnetic resonance imaging (mri) of relapsing-remitting multiple sclerosis. In *2017 12th International Workshop on Self-Organizing Maps and Learning Vector Quantization, Clustering and Data Visualization (WSOM)*, pages 1–7, June 2017. 106

[167] M. Abadi, S. Jovanovic, K. Ben Khalifa, S. Weber, and M. H. Bedoui. A hardware configurable self-organizing map for real-time color quantization. In *2016 IEEE International Conference on Electronics, Circuits and Systems (ICECS)*, pages 336–339, Dec 2016. 121, 122

[168] M. Abadi, S. Jovanovic, K. Ben-Khalifa, S. Weber, and M.H. Bedoui. A scalable and adaptable hardware noc-based self organizing map. *Microprocessors and Microsystems*, 57 :1 – 14, 2018. 121, 122

[169] S. Orts, J. Garcia-Rodriguez, D. Viejo, M. Cazorla, and V. Morell. Gpgpu implementation of growing neural gas : Application to 3d scene reconstruction. *Journal of Parallel and Distributed Computing*, 72(10) :1361 – 1372, 2012. 125

[170] S. Matharage, D. Alahakoon, J. Rajapakse, and P. Huang. Fast growing self organizing map for text clustering. In Bao-Liang Lu, Liqing Zhang, and James Kwok, editors, *Neural Information Processing*, pages 406–415, Berlin, Heidelberg, 2011. Springer Berlin Heidelberg. 125, 126

[171] M. Cirrincione, M. C. Di Piazza, M. Pucci, and G. Vitale. Real-time simulation of photovoltaic arrays by growing neural gas controlled dc-dc converter. In *2008 IEEE Power Electronics Specialists Conference*, pages 2004–2010, June 2008. 125

[172] Y. Cui, Y. Zhang, and Y. Cui. A growing neural gas (gng) model for hand gesture recognition. In *2013 Ninth International Conference on Natural Computation (ICNC)*, pages 232–237, July 2013. 125

[173] H. Liu, M. Kurihara, S. Oyama, and H. Sato. An incremental self-organizing neural network based on enhanced competitive hebbian learning. In *The 2013 International Joint Conference on Neural Networks (IJCNN)*, pages 1–8, Aug 2013. 125

[174] S. Y. Huang and R. H. Tsaih. The prediction approach with growing hierarchical self-organizing map. In *The 2012 International Joint Conference on Neural Networks (IJCNN)*, pages 1–7, June 2012. 125, 127

[175] T. Ayadi, T.M. Hamdani, and A.M. Alimi. Migsom : Multilevel interior growing self-organizing maps for high dimensional data clustering. *Neural Processing Letters*, 36(3) :235–256, Dec 2012. 125, 127, 128

[176] V. Moreli, M. Cazorla, S. Orts-Escolano, and J. García-Rodríguez. 3d maps representation using gng. In *2014 International Joint Conference on Neural Networks (IJCNN)*, pages 1482–1487, July 2014. 125

[177] A.F. R. Araujo and R.L. M. E. Rego. Self-organizing maps with a time-varying structure. *ACM Comput. Surv.*, 46 :7 :1–7 :38, July 2013. 125, 126

[178] T. Martinetz and K. Schulten. Topology representing networks. *Neural Networks*, 7(3) :507 – 522, 1994. 125

[179] A. Vathy-Fogarassy, A. Werner-Stark, B. Gal, and J. Abonyi. Visualization of topology representing networks. In Hujun Yin, Peter Tino, Emilio Corchado, Will Byrne, and Xin Yao, editors, *Intelligent Data Engineering and Automated Learning - IDEAL 2007*, pages 557–566, Berlin, Heidelberg, 2007. Springer Berlin Heidelberg. 125

[180] A. Vathy-Fogarassy, A. Kiss, and J. Abonyi. *Topology Representing Network Map – A New Tool for Visualization of High-Dimensional Data*, pages 61–84. Springer Berlin Heidelberg, Berlin, Heidelberg, 2008. 125

[181] R. H. White. Competitive hebbian learning : Algorithm and demonstrations. *Neural Networks*, 5(2) :261–275, 1992. 125

[182] B. Fritzke. A growing neural gas network learns topologies. In *Proceedings of the 7th International Conference on Neural Information Processing Systems*, NIPS'94, pages 625–632, Cambridge, MA, USA, 1995. MIT Press. 125, 127

[183] T. MARTINETZ and K. Scrwrxrss. A "neural-gas" network learns topologies. *Artificial Neural Network*, 2 :397–402, 1991. 125

[184] S. Marsland, J. Shapiro, and U. Nehmzow. A self-organising network that grows when required. *Neural Networks*, 15(8) :1041 – 1058, 2002. 126

[185] A. Kalos. Automated heuristic growing of neural networks for nonlinear time series models. In *Proceedings. 2005 IEEE International Joint Conference on Neural Networks, 2005.*, volume 1, pages 320–325 vol. 1, July 2005. 126

[186] G. Cheng, Z. Song, J. Yang, and R. Gao. On growing self - organizing neural networks without fixed dimensionality. In *2006 International Conference on Computational Inteligence for Modelling Control and Automation and International Conference on Intelligent*

Agents Web Technologies and International Commerce (CIMCA'06), pages 164–164, Nov 2006. 126

[187] S. Yoshida and N. Kubota. Growing neural gas based conversation selection model for robot partner and human communication system. In *2014 IEEE Symposium on Robotic Intelligence in Informationally Structured Space (RiiSS)*, pages 1–5, Dec 2014. 126

[188] D. Fiser, J. Faigl, and M. Kulich. Growing neural gas efficiently. *Neurocomputing*, 104 :72 – 82, 2013. 126

[189] M. Cirrincione, M. Pucci, and G. Vitale. Growing neural gas (gng)-based maximum power point tracking for high-performance wind generator with an induction machine. *IEEE Transactions on Industry Applications*, 47(2) :861–872, March 2011. 126

[190] M. Dittenbach, A. Rauber, and D. Merkl. Uncovering hierarchical structure in data using the growing hierarchical self-organizing map. *Neurocomputing*, 48(1) :199 – 216, 2002. 126

[191] S. Bizzi, R. F. Harrison, and D. N. Lerner. The growing hierarchical self-organizing map (ghsom) for analysing multi-dimensional stream habitat datasets. In *18th World IMACS / MODSIM Congress, Cairns, Australia*, pages 734–740, 2009. 126

[192] E. J. Palomo and E. López-Rubio. The growing hierarchical neural gas self-organizing neural network. *IEEE Transactions on Neural Networks and Learning Systems*, 28(9) :2000–2009, Sept 2017. 126

[193] Y. Toda and N. Kubota. Feature extraction based on hierarchical growing neural gas for informationally structured space. In *The 2013 International Joint Conference on Neural Networks (IJCNN)*, pages 1–7, Aug 2013. 126

[194] X. Qiang, G. Cheng, and Z. Li. A survey of some classic self-organizing maps with incremental learning. In *2010 2nd International Conference on Signal Processing Systems*, volume 1, pages V1–804–V1–809, July 2010. 126

[195] R. L. M. E. Rego, A. F. R. Araujo, and F. B. de Lima Neto. Growing self-reconstruction maps. *IEEE Transactions on Neural Networks*, 21(2) :211–223, Feb 2010. 126, 127

[196] J. Blackmore and R. Miikkulainen. Visualizing high-dimensional structure with the incremental grid growing neural network. Master's thesis, Department of Computer Sciences, The University of Texas at Austin, Austin, TX, 1995. Technical Report AI95-238. 127

[197] R. L. M. E. do Rêgo, P. H. M. Ferreira, and A. F. R. Araújo. Reconstructing anatomical structures with growing self-reconstruction maps. In *2011 IEEE International Conference on Systems, Man, and Cybernetics*, pages 2976–2981, Oct 2011. 127

Résumé

Depuis son introduction en *1982*, la carte auto-organisatrice de Kohonen (*Self-Organizing Map : SOM*) a prouvé ses capacités de classification et visualisation des données multidimensionnelles dans différents domaines d'application. Les implémentations matérielles de la carte *SOM*, en exploitant le taux de parallélisme élevé de l'algorithme de Kohonen, permettent d'augmenter les performances de ce modèle neuronal souvent au détriment de la flexibilité. D'autre part, la flexibilité est offerte par les implémentations logicielles qui quant à elles ne sont pas adaptées pour les application temps réel à cause de leurs performances temporelles limitées. Dans cette thèse nous avons proposé une architecture matérielle distribuée, adaptable, flexible et extensible de la carte *SOM* à base de *NoC* dédiée pour une implantation matérielle sur *FPGA*. A base de cette approche, nous avons également proposé une architecture matérielle innovante d'une carte *SOM* à structure croissante au cours de la phase d'apprentissage.

Mots-clés: Réseau de neurones artificiels (*RNA*), Carte auto-organisatrice (*SOM*), Réseau sur puce (*NoC*), Multiprocesseurs système sur puce (*MPSoC*), Architecture auto-adaptable, Réseau de neurones évolutif

Abstract

Since its introduction in *1982*, Kohonen's Self-Organizing Map (*SOM*) showed its ability to classify and visualize multidimensional data in various application fields. Hardware implementations of *SOM*, by exploiting the inherent parallelism of the Kohonen algorithm, allow to increase the overall performances of this neuronal network, often at the expense of the flexibility. On the other hand, the flexibility is offered by software implementations which on their side are not suited for real-time applications due to the limited time performances. In this thesis we proposed a distributed, adaptable, flexible and scalable hardware architecture of *SOM* based on Network-on-Chip (*NoC*) designed for *FPGA* implementation. Moreover, based on this approach we also proposed a novel hardware architecture of a growing *SOM* able to evolve its own structure during the learning phase.

Keywords: Artificial neural network (*ANN*), Self-Organizing Maps (*SOM*), Network-on-Chip (*NoC*), multiprocessor system-on-chip (*MPSoC*), Self-adaptive architecture, Evolving Artificial Neural Networks