(19) RÉPUBLIQUE FRANÇAISE

INSTITUT NATIONAL DE LA PROPRIÉTÉ INDUSTRIELLE

COURBEVOIE

(11) **N° de publication :** **3 091 389**

(à n'utiliser que pour les commandes de reproduction)

(21) **N° d'enregistrement national :** **19 06119**

(51) Int Cl[8] : ***G 06 N 3/06*** *(2019.01), G 06 F 15/76, 9/30, 9/46, G 06 N 3/08*

(12) DEMANDE DE BREVET D'INVENTION A1

(22) **Date de dépôt :** 07.06.19.

(30) **Priorité :** 31.12.18 GB 1821301.7.

(43) **Date de mise à la disposition du public de la demande :** 03.07.20 Bulletin 20/27.

(56) **Liste des documents cités dans le rapport de recherche préliminaire :** *Ce dernier n'a pas été établi à la date de publication de la demande.*

(60) **Références à d'autres documents nationaux apparentés :**

◯ **Demande(s) d'extension :**

(71) **Demandeur(s) :** *Graphcore Limited Société régie par les lois de l'Angleterre et du Pays de Galles* — GB.

(72) **Inventeur(s) :** ALEXANDER Alan Graham, KNOWLES Simon Christian et GORE Mrudula.

(73) **Titulaire(s) :** Graphcore Limited Société régie par les lois de l'Angleterre et du Pays de Galles.

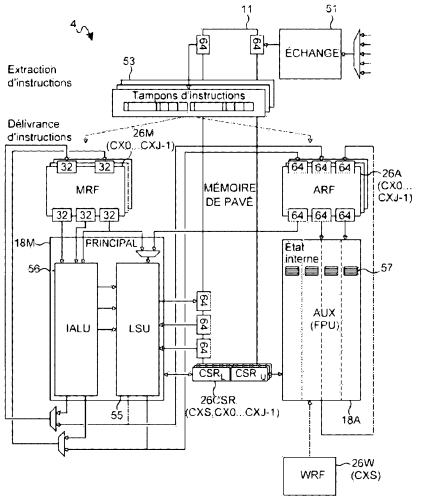
(74) **Mandataire(s) :** CABINET BEAUMONT.

(54) BANCS DE REGISTRES DANS UN PROCESSEUR À FILS D'EXÉCUTION MULTIPLES.

(57) BANCS DE REGISTRES DANS UN PROCESSEUR À FILS D'EXÉCUTION MULTIPLES

Processeur comprenant une unité d'exécution à fils d'exécution en barillet pour exécuter des fils d'exécution simultanés, et un ou plusieurs bancs de registres comprenant un ensemble respectif de registres de contexte pour chaque fil simultané. L'un des bancs de registres comprend en outre un ensemble de registres de poids partagés commun à certains ou à la totalité des fils d'exécution simultanés. Les types d'instructions définis dans le jeu d'instructions du processeur comprennent une instruction arithmétique ayant des opérandes spécifiant une source et une destination parmi un ensemble respectif de registres arithmétiques du fil dans lequel l'instruction arithmétique est exécutée. L'unité d'exécution est agencée de manière à, en réponse au code opération de l'instruction arithmétique, réaliser une opération comprenant la multiplication d'une entrée provenant de la source par au moins l'un des poids provenant d'au moins l'un des registres de poids partagés, et placer un résultat dans la destination.

Figure pour l'abrégé : Fig. 4



FR 3 091 389 - A1

Description

Titre de l'invention : BANCS DE REGISTRES DANS UN PROCESSEUR À FILS D'EXÉCUTION MULTIPLES

Domaine technique

[0001] La présente description concerne un agencement de bancs de registres destiné à être utilisé par des instructions de code machine exécutées dans un processeur à fils d'exécution multiples.

Technique antérieure

[0002] On a trouvé de plus en plus d'intérêt à développer des processeurs conçus pour des applications spécifiques, comme des unités de traitement graphique (GPU) et des processeurs de signal numérique (DSP). Un autre type de processeur spécifique à des applications auquel on a récemment porté de plus en plus d'intérêt est un type dédié à des applications d'intelligence artificielle, dénommé par la demanderesse "IPU" (de l'anglais Intelligence Processing Unit - unité de traitement pour intelligence). Ces processeurs peuvent être utilisés par exemple comme processeurs accélérateurs agencés pour réaliser des tâches allouées par un hôte, comme par exemple pour réaliser un apprentissage ou pour aider à l'apprentissage d'un modèle de connaissance comme un réseau neuronal, ou pour réaliser ou aider à réaliser des prédictions ou des inférences sur la base d'un tel modèle.

[0003] Un algorithme d'intelligence artificielle est basé sur la réalisation itérative de mises à jour d'un "modèle de connaissance", qui peut être représenté par un graphe de multiples nœuds interconnectés. Chaque nœud représente une fonction de ses entrées. Certains nœuds reçoivent les entrées du graphe et certains nœuds reçoivent des entrées provenant d'un ou plusieurs autres nœuds, tandis que les sorties de certains nœuds constituent les entrées d'autres nœuds, et les sorties de certains nœuds fournissent la sortie du graphe (et dans certains cas un nœud donné peut même comporter tout cela : des entrées du graphe, des sorties du graphe et des connexions à d'autres nœuds). En outre, la fonction au niveau de chaque nœud est paramétrée par un ou plusieurs paramètres respectifs, par exemple des poids. Pendant une étape d'apprentissage le but est, sur la base d'un ensemble de données d'entrée expérimentales, de trouver des valeurs pour les divers paramètres telles que le graphe dans son ensemble va générer une sortie souhaitée pour une plage d'entrées possibles. Divers algorithmes pour réaliser cela sont connus dans la technique, comme un algorithme à rétro-programmation basé sur une descente de gradient stochastique. Sur de multiples itérations basées sur les données d'entrée, les paramètres sont progressivement ajustés pour diminuer leurs erreurs, et ainsi le graphe converge vers une solution. Dans une

étape ultérieure, le modèle appris peut alors être utilisé pour faire des prédictions de sorties étant donné un ensemble spécifié d'entrées ou pour réaliser des inférences quant aux entrées (causes) étant donné un ensemble spécifié de sorties.

[0004] Un processeur conçu pour des applications d'intelligence artificielle peut inclure dans son jeu d'instructions des instructions dédiées pour réaliser des opérations arithmétiques utilisées communément dans les applications d'intelligence artificielle (le jeu d'instructions étant le jeu fondamental de types d'instructions de code machine que l'unité d'exécution du processeur peut reconnaitre, chaque type étant défini par un code opération respectif et par zéro, ou plus, opérandes). Par exemple, une opération courante nécessaire dans les applications d'intelligence artificielle comme les réseaux neuronaux est la convolution d'un noyau avec un ensemble de données d'entrée, le noyau représentant les poids d'un nœud dans le réseau neuronal. Pour réaliser la convolution d'un noyau d'une taille significative avec les données, la convolution peut être décomposée en multiples produits vectoriels ou matriciels produisant chacun en sortie une somme partielle à accumuler avec la sortie de produits ultérieurs. Il existe déjà des processeurs qui incluent dans leur jeu d'instructions des instructions arithmétiques dédiées à la réalisation d'opérations du type multiplication vectorielle et matricielle destinées à être utilisées dans la réalisation de convolutions.

Résumé de l'invention

[0005] Un processeur peut aussi assurer un soutien pour l'exécution simultanée de multiples fils d'exécution de programme. Ce soutien comprend typiquement un ensemble respectif de registres de contexte pour chaque fil d'exécution, chaque ensemble étant utilisé pour contenir l'état de programme (le "contexte") de l'un respectif d'une pluralité de fils devant être exécutés simultanément. L'état de programme contenu dans les registres de contexte d'un fil donné comprend typiquement un compteur de programme pour ce fil, un ou plusieurs registres d'état de commande enregistrant un état du fil (par exemple en pause, en exécution, etc.), et une pluralité de registres d'opérandes comprenant des registres d'opérandes arithmétiques destinés à contenir temporairement des valeurs sur lesquelles on doit opérer et qui sont fournies par des instructions arithmétiques du fil d'exécution respectif. Les différents fils simultanés sont entrelacés dans le temps par l'intermédiaire d'un pipeline d'exécution commun dans des créneaux temporels d'exécution respectifs différents, chaque créneau utilisant seulement les registres d'un ensemble respectif différent parmi les ensembles de registres de contexte.

[0006] Habituellement, chaque fil comprend son propre ensemble respectif de registres d'opérandes arithmétiques dans un banc de registres arithmétiques séparé pour chaque fil. Lorsqu'une instruction est exécutée en tant que partie d'un fil donné, il est implicite

qu'elle utilise toujours les registres d'opérandes arithmétiques se trouvant dans le banc de registres arithmétiques de ce fil particulier.

[0007] On a identifié ici qu'il peut se trouver des applications dans lesquelles des fils d'exécution peuvent en fait opérer sur certaines valeurs d'opérandes qui sont les mêmes et aussi sur certains autres opérandes qui sont spécifiques au fil individuel. Un exemple pourrait se trouver dans une multiplication vectorielle ou matricielle réalisée en tant que partie d'une convolution, dans laquelle l'instruction dans chaque fil multiplie un ensemble commun de poids (partagé entre des fils) par des données d'entrées qui sont spécifiques au fil individuel. Un exemple particulier de cela peut survenir dans un réseau neuronal à convolution où de nombreux nœuds comprennent en fait les mêmes poids mais avec des connexions différentes. Considérons par exemple un scénario dans lequel chaque fil d'exécution est agencé pour réaliser le traitement d'un nœud respectif différent dans le réseau neuronal, comme la convolution d'un noyau commun de poids avec des données d'entrée respectives pour détecter un certain élément caractéristique. Dans de tels scénarios, il serait avantageux de prévoir un mécanisme permettant à une instruction arithmétique donnée d'opérer sur une combinaison d'un ou plusieurs opérandes de poids communs partagés entre des fils et d'un ou plusieurs opérandes spécifiques aux fils respectifs individuels. Ce problème n'est en aucun cas spécifique de réseaux neuronaux et pourrait apparaître dans toute application qui se retrouve à utiliser certaines valeurs d'opérandes partagées et certains opérandes spécifiques à des fils.

[0008] Selon un aspect décrit ici, on prévoit un processeur comprenant : un ou plusieurs bancs de registres ; et une unité d'exécution agencée pour exécuter des instances de types d'instructions définis dans un jeu d'instructions, chaque instruction se trouvant dans le jeu d'instructions étant constituée d'un code opération et d'un ou plusieurs opérandes. L'unité d'exécution est une unité d'exécution à fils en barillet agencée pour exécuter une pluralité de fils simultanés chacun dans un créneau respectif différent d'une séquence répétitive de créneaux temporels entrelacés, et pour chacun des fils simultanés, lesdits un ou plusieurs bancs de registres comprennent un ensemble respectif de registres de contexte agencé pour contenir un état de programme du fil respectif, chaque ensemble de registres de contexte comprenant un ensemble respectif de registres d'opérandes arithmétiques destiné à être utilisés par le fil respectif. L'un desdits un ou plusieurs bancs de registres comprend en outre un ensemble de registres de poids partagés commun à certains ou à la totalité des fils simultanés. Les types d'instructions définis dans le jeu d'instructions comprennent une instruction arithmétique comportant des opérandes spécifiant une source et une destination dans l'ensemble respectif de registres arithmétiques du fil dans lequel l'instruction arithmétique est exécutée. L'unité d'exécution est agencée afin de, en réponse au code

opération de l'instruction arithmétique, réaliser une opération comprenant la multiplication d'une entrée provenant de ladite source par au moins l'un des poids provenant d'au moins l'un des registres de poids partagés, et placer un résultat dans la destination.

[0009] Dans des modes de réalisation, ledit au moins l'un des registres de poids partagés peut être implicite d'après le code opération de l'instruction arithmétique, n'étant spécifié par aucun d'opérande de l'instruction arithmétique.

[0010] Dans des modes de réalisation, l'instruction arithmétique peut prendre un autre opérande spécifiant ledit au moins l'un des registres de poids partagés parmi l'ensemble de registres de poids partagés.

[0011] Dans des modes de réalisation, l'entrée peut comprendre un vecteur, et la multiplication peut comprendre un produit scalaire de l'entrée par un vecteur de poids provenant des registres de poids partagés.

[0012] Dans des modes de réalisation, ledit au moins l'un des registres de poids partagés peut comprendre un sous-ensemble des registres de poids partagés parmi une pluralité de sous-ensembles, chaque sous-ensemble contenant un vecteur de poids respectif ; et l'autre opérande peut sélectionner à partir duquel sous-ensemble il faut prendre le vecteur de poids à utiliser dans la multiplication.

[0013] Dans des modes de réalisation, l'instruction arithmétique peut comprendre l'une des instructions suivantes : une instruction de produit scalaire vectoriel, une instruction de produit scalaire vectoriel à accumulation, une instruction de produit matriciel, une instruction de produit matriciel à accumulation, ou une instruction de convolution.

[0014] Dans des modes de réalisation, les fils simultanés peuvent comprendre une pluralité de fils de travail et l'unité d'exécution peut être en outre agencée pour exécuter, au moins à certains instants, un sous-programme superviseur comprenant au moins un fil superviseur agencé pour gérer les fils de travail.

[0015] Dans des modes de réalisation, le sous-programme superviseur peut être agencé pour écrire les poids dans le banc de registres de poids partagés.

[0016] Dans des modes de réalisation, les poids se trouvant dans les registres de poids partagés peuvent être écrits seulement par le sous-programme superviseur, et les fils de travail peuvent être seulement capables de lire les registres de poids partagés.

[0017] Dans des modes de réalisation, les registres de contexte peuvent comprendre l'un respectif des ensembles de registres de contexte pour chacun des fils de travail qui peuvent être exécutés simultanément, et un ensemble additionnel de registres de contexte peut être agencé pour contenir un état de programme du sous-programme superviseur.

[0018] Dans des modes de réalisation, le sous-programme superviseur peut être agencé pour commencer en s'exécutant initialement dans tous les créneaux, et pour écrire les poids

avant de lancer les fils de travail ; le sous-programme superviseur pouvant lancer chacun des fils de travail en abandonnant certains ou la totalité des créneaux dans lesquels le sous-programme superviseur s'exécute initialement à certains fils respectifs parmi les fils de travail.

[0019] Dans des modes de réalisation, le jeu d'instructions peut inclure une instruction d'exécution qui, lorsqu'elle est exécutée en tant que partie du sous-programme superviseur, amène le créneau dans lequel l'instruction d'exécution est exécutée à être abandonné à l'un des fils de travail de sorte que le fil de travail est lancé dans ce créneau à la place du sous-programme superviseur.

[0020] Dans des modes de réalisation, le jeu d'instructions peut inclure une instruction de sortie qui, lorsqu'elle est exécutée en tant que partie de l'un des fils de travail, amène le créneau dans lequel l'instruction de sortie est exécutée à être restitué au sous-programme superviseur de sorte que le sous-programme superviseur continue à s'exécuter dans ce créneau de nouveau à la place du fil de travail.

[0021] Dans des modes de réalisation, les bancs de registres comprennent un banc de registres arithmétiques séparé pour chaque fil de travail simultané, le banc de registres arithmétiques respectif comprenant les registres d'opérandes arithmétiques respectifs.

[0022] Dans des modes de réalisation, les bancs de registres peuvent comprendre un banc de registres de poids séparé comprenant les registres de poids.

[0023] Dans des modes de réalisation, le banc de registres de poids peut être agencé de telle sorte qu'il peut être écrit seulement par le sous-programme superviseur et les fils de travail peuvent seulement lire le banc de registres de poids.

[0024] Selon un autre aspect de la présente invention, on prévoit un procédé d'actionnement d'un processeur agencé selon l'un quelconque des modes de réalisation mentionnés ci-avant ou ailleurs ici, le procédé comprenant l'exécution d'un programme comprenant une ou plusieurs instances de l'instruction arithmétique sur le processeur par l'intermédiaire de l'unité d'exécution.

Brève description des dessins

[0025] Pour faciliter la compréhension de modes de réalisation de la présente description et pour montrer comment de tels modes de réalisation peuvent être mis en pratique, on va faire référence, seulement à titre d'exemple, aux dessins joints dans lesquels :

[0026] [fig.1] la figure 1 est un schéma blocs d'un exemple de processeur à fils d'exécution multiples,

[0027] [fig.2] la figure 2 représente schématiquement un schéma de créneaux temporels entrelacés,

[0028] [fig.3] la figure 3 représente schématiquement un fil superviseur et une pluralité de fils de travail s'exécutant dans une pluralité de créneaux temporels entrelacés,

[0029] [fig.4] la figure 4 représente schématiquement une structure de blocs logiques d'un exemple de processeur,

[0030] [fig.5] la figure 5 est un schéma blocs d'un processeur comprenant une matrice de processeurs constituants,

[0031] [fig.6] la figure 6 représente schématiquement un graphe utilisé dans un algorithme d'intelligence artificielle,

[0032] [fig.7] la figure 7 représente schématiquement un schéma de disposition d'adresses destiné à être utilisé dans la mise en œuvre d'un type d'instruction de chargement-stockage,

[0033] [fig.8] la figure 8 représente schématiquement un agencement de valeurs de pas de progression prédéterminées dans un ensemble de registres de pas,

[0034] [fig.9] la figure 9 représente schématiquement une convolution d'un noyau 3D K avec un volume de données d'entrée,

[0035] [fig.10] la figure 10 représente schématiquement une multiplication matricielle réalisée par une séquence de phases d'une instruction de produit matriciel à accumulation,

[0036] [fig.11] la figure 11 représente plus précisément le fonctionnement des instructions de produit matriciel à accumulation,

[0037] [fig.12] la figure 12 donne un exemple d'une série de boucles d'une séquence d'instructions de produit matriciel à accumulation agencée pour réaliser une convolution,

[0038] [fig.13] la figure 13 représente schématiquement le fonctionnement d'une instruction de convolution, et

[0039] [fig.14] la figure 14 donne un exemple d'une série d'instructions de convolution.

Description des modes de réalisation

[0040] La figure 1 illustre un exemple de processeur 4 selon des modes de réalisation de la présente description. Le processeur 4 comprend une unité de traitement à fils d'exécution multiples 10 prenant la forme d'une unité de traitement à fils en barillet, et une mémoire locale 11 (c'est-à-dire sur le même pavé dans le cas d'une matrice de pavés multiples, ou sur la même puce dans le cas d'une puce à un seul processeur). Une unité de traitement à fils d'en barillet est un type d'unité de traitement à fils d'exécution multiples dans lequel le temps d'exécution du pipeline est divisé en une séquence répétitive de créneaux temporels entrelacés, chacun d'eux pouvant être occupé par un fil d'exécution donné. On peut aussi appeler cela exécution simultanée, décrit plus en détail immédiatement dans la suite. La mémoire 11 comprend une mémoire d'instructions 12 et une mémoire de données 22 (qui peuvent être mises en œuvre dans des modules mémoire adressables différents ou dans des régions dif-

férentes du même module mémoire adressable). La mémoire d'instructions 12 mémorise du code machine à exécuter par l'unité de traitement 10, tandis que la mémoire de données 22 mémorise à la fois des données sur lesquelles va opérer le code exécuté et des données de sortie produites par le code exécuté (par exemple en résultat de telles opérations).

[0041] La mémoire 12 mémorise une pluralité de fils d'exécution différents d'un programme, chaque fil d'exécution comprenant une séquence respective d'instructions pour réaliser une certaine tâche ou certaines tâches. On notera qu'une instruction telle que mentionnée ici désigne une instruction de code machine, c'est-à-dire une instance d'une des instructions fondamentales du jeu d'instructions du processeur, constituée d'un seul code opération et de zéro, ou plus, opérandes. Dans des modes de réalisation, le programme comprend une pluralité de fils de travail, et un sous-programme superviseur qui peut être structuré sous forme d'un ou plusieurs fils superviseurs. Cela va être décrit plus en détail immédiatement dans la suite.

[0042] Un processeur à fils d'exécution multiples est un processeur capable d'exécuter de multiples fils de programme les uns à côté des autres, typiquement de manière simultanée. Une exécution simultanée signifie que les fils partagent un pipeline d'exécution commun (ou au moins une partie commune d'un pipeline) et que des fils différents sont entrelacés dans ce même pipeline à exécution partagée dans différents créneaux temporels entrelacés dans un cycle répétitif. Cela augmente les performances en raison de plus nombreuses opportunités pour cacher la latence du pipeline. Le processeur comprend du matériel qui est commun aux multiples fils d'exécution différents (par exemple une mémoire d'instructions, une mémoire de données et/ou une unité d'exécution en commun) ; mais pour prendre en charge le fonctionnement à fils multiples, le processeur comprend aussi du matériel dédié spécifique à chaque fil.

[0043] Le matériel dédié comprend un ensemble séparé de registres de contexte 26 pour au moins chacun des fils qui peuvent être exécutés simultanément, c'est-à-dire un ensemble par créneau dans le cycle. Le mot "contexte", lorsqu'on parle de processeur à fils d'exécution multiples, fait référence à l'état de programme de l'un respectif des fils en cours d'exécution côte à côte (par exemple valeur de compteur de programme, valeurs de statut et d'opérandes courants). Les registres de contexte font référence aux registres respectifs destinés à représenter cet état de programme du fil respectif. Les registres se trouvant dans un banc de registres sont distincts de la mémoire générale en ce que les adresses des registres sont fixées sous forme de bits dans des mots d'instructions, alors que les adresses mémoire peuvent être calculées en exécutant des instructions.

[0044] Dans l'unité de traitement 10, de multiples fils différents parmi les fils provenant de la mémoire d'instructions 12 peuvent être entrelacés dans un seul pipeline d'exécution

13 (bien que typiquement seul un sous-ensemble de la totalité des fils mémorisés dans la mémoire d'instructions peut être entrelacé à un point donné dans le programme global). L'unité de traitement à fils multiples 10 comprend une pluralité d'ensembles de registres de contexte 26, chaque ensemble 26 étant agencé pour représenter l'état (contexte) d'un fil respectif différent parmi les fils qui peuvent être exécutés simultanément. L'unité de traitement à fils d'exécution multiples 10 comprend aussi un pipeline d'exécution partagé 13, qui est commun aux fils exécutés simultanément, et un ordonnanceur 24 pour planifier les fils d'exécution simultanés pour une exécution à travers le pipeline partagé de manière entrelacée, par exemple à tour de rôle. L'unité de traitement 10 est connectée à une mémoire d'instructions partagée 12 commune à la pluralité de fils, et à une mémoire de données partagée 22 qui est ici encore commune à la pluralité de fils.

[0045] Le pipeline d'exécution 13 comprend un étage d'extraction 14, un étage de décodage 16 et un étage d'exécution 18 comprenant une unité d'exécution qui peut réaliser des opérations arithmétiques et logiques, des calculs d'adresses, des opérations de chargement et de stockage, et d'autres opérations, comme définies par l'architecture du jeu d'instructions.

[0046] Chaque ensemble de registres de contexte 26 comprend un ou plusieurs registres de commande respectifs comprenant au moins un compteur de programme (PC) pour le fil respectif (pour garder la trace de l'adresse d'instruction à laquelle le fils s'exécute actuellement), et dans des modes de réalisation aussi un ensemble d'un ou plusieurs registres d'état de commande (CSR) enregistrant un statut courant du fil respectif (comme par exemple s'il est actuellement en exécution ou en pause). Chaque ensemble de bancs de registres de contexte 26 comprend aussi un ensemble respectif de registres d'opérandes, pour contenir temporairement des opérandes des instructions exécutées par le fil respectif, c'est-à-dire des valeurs sur lesquelles on opère ou qui résultent d'opérations définies par les codes d'opération des instructions du fil respectif lorsqu'elles sont exécutées. Chaque ensemble de registres 26 peut être mis en œuvre dans un ou plusieurs bancs de registres.

[0047] L'étage d'extraction 14 a accès au compteur de programme (PC) de chacun des contextes. Pour chaque fil respectif, l'étage d'extraction 14 extrait l'instruction suivante de ce fil de l'adresse suivante dans la mémoire de programme 12 comme indiqué par le compteur de programme. Le compteur de programme s'incrémente automatiquement à chaque cycle d'exécution à moins qu'il ne soit dérouté par une instruction de branchement. L'étage d'extraction 14 passe ensuite l'instruction extraite à l'étage de décodage 16 pour qu'elle soit décodée, et l'étage de décodage 16 passe ensuite une indication de l'instruction décodée à l'unité d'exécution 18 accompagnée des adresses décodées des éventuels registres d'opérandes qui sont spécifiés dans

l'instruction, afin que l'instruction soit exécutée. L'unité d'exécution 18 a accès aux registres d'opérandes et aux registres d'état de commande, qu'elle peut utiliser dans l'exécution de l'instruction sur la base des adresses de registres décodées, comme dans le cas d'une instruction arithmétique (par exemple en additionnant, en multipliant, en soustrayant ou en divisant les valeurs se trouvant dans deux registres d'opérandes et en fournissant le résultat à un autre registre d'opérande du fil respectif). Ou bien si l'instruction définit un accès à la mémoire (chargement ou stockage), la logique de chargement/stockage de l'unité d'exécution 18 charge une valeur à partir de la mémoire de données dans un registre d'opérande du fil respectif, ou mémorise une valeur provenant d'un registre d'opérande du fil respectif dans la mémoire de données 22, conformément à l'instruction.

[0048] L'étage d'extraction 14 est connecté de manière à extraire des instructions à exécuter à partir de la mémoire d'instructions 12, sous le contrôle de l'ordonnanceur 24. L'ordonnanceur 24 est agencé pour contrôler l'étage d'extraction 14 pour extraire une instruction à partir de chacun d'un ensemble de fils s'exécutant simultanément tour à tour dans une séquence répétitive de créneaux temporels, divisant ainsi les ressources du pipeline 13 en une pluralité de créneaux temporels entrelacés dans le temps, comme on va le décrire plus en détail immédiatement dans la suite. Par exemple, le schéma d'ordonnancement pourrait être un tour de rôle ou un tour de rôle pondéré. Un autre terme pour désigner un processeur opérant de cette manière est processeur à fils d'exécution en barillet.

[0049] Un exemple du schéma d'entrelacement mis en œuvre par l'ordonnanceur 24 est illustré en figure 2. Ici, les fils simultanés sont entrelacés selon un schéma de tour de rôle dans lequel, dans chaque tour du schéma, le tour est divisé en une séquence de créneaux temporels S0, S1, S2…SJ-1 (par exemple J=4, ou J=6), chaque créneau étant destiné à exécuter un fil respectif. Typiquement chaque créneau a une longueur égale à un cycle d'exécution et les différents créneaux sont de taille égale, bien que cela ne soit pas nécessairement ainsi dans tous les modes de réalisation possibles, par exemple un schéma de tour de rôle pondéré est aussi possible, schéma dans lequel certains fils obtiennent plus de cycles que d'autres par tour d'exécution. En général l'exécution de fils en barillet peut utiliser un ordonnancement soit à tour de rôle régulier soit à tour de rôle pondéré, dans ce dernier cas la pondération pouvant être fixe ou adaptative.

[0050] Quelle que soit la séquence pour chaque tour d'exécution, ce motif se répète ensuite, chaque tour comprenant une instance respective de chacun des créneaux temporels. On notera par conséquent qu'un créneau temporel auquel on fait référence ici désigne l'emplacement alloué de manière répétitive dans la séquence, et non une instance particulière du créneau temporel dans une répétition donnée de la séquence. Dit d'une autre manière, l'ordonnanceur 24 répartit les cycles d'exécution du pipeline 13 dans

une pluralité de canaux d'exécution entrelacés dans le temps (multiplexés par séparation temporelle), chacun comprenant une récurrence d'un créneau temporel respectif dans une séquence répétitive de créneaux temporels. Dans le mode de réalisation illustré, il y a quatre créneaux temporels, mais cela n'est que dans un but illustratif et d'autres nombres sont possibles. Par exemple dans un mode de réalisation préféré, il y a en fait six créneaux temporels.

[0051] Dans des modes de réalisation, les registres de contexte 26 comprennent un ensemble respectif de registres de contexte de travail CX0… CX(J-1) pour chacun des J fils qui peuvent être exécutés simultanément (J=3 dans l'exemple illustré mais cela n'est pas limitatif), et un banc de registres de contexte de superviseur additionnel, CXS. Les bancs de registres de contexte de travail contiennent les contextes de fils de travail, et le banc de registres de contexte de superviseur contient le contexte d'un fil superviseur. On notera que dans des modes de réalisation le contexte de superviseur comporte un nombre de registres différent de celui de chacun des fils de travail. L'unité de traitement 10 comprend ainsi un banc de registres de contexte 26 de plus qu'il y a de créneaux temporels, c'est-à-dire qu'elle supporte un contexte de plus que le nombre de créneaux temporels entrelacés qu'elle peut gérer pour des fils en barillet.

[0052] Chacun des contextes de travail CX0…CXJ est utilisé pour représenter l'état de l'un respectif d'une pluralité de fils de travail affectés actuellement à l'un des quatre créneaux temporels d'exécution S0…SJ, pour réaliser n'importe quelles tâches de calcul spécifiques d'application que souhaite le programmeur (on notera de nouveau que cela peut seulement être un sous-ensemble du nombre total de fils de travail du programme mémorisé dans la mémoire d'instructions 12). Le contexte additionnel CXS est utilisé pour représenter l'état d'un "fil superviseur" (SV) dont le rôle est de coordonner l'exécution des fils de travail, au moins dans le sens de l'affectation de celui des fils de travail W qui doit être exécuté dans tel ou tel créneau des créneaux temporels S0, S1, S2... et à quel point dans le programme global. Optionnellement, le fil superviseur peut avoir d'autres responsabilités de "superviseur" ou de coordination, comme la réalisation d'échanges externes ou de synchronisations à barrières. On notera bien sûr que le cas illustré de J=4 n'est qu'un exemple de mise en œuvre dans un but illustratif. Par exemple, dans une autre mise en œuvre J=6 (six créneaux temporels, six contextes de travail et un contexte de superviseur).

[0053] En référence à la figure 3, dans des modes de réalisation le fil superviseur SV ne possède pas son propre créneau par lui-même dans le schéma de créneaux temporels d'exécution entrelacés. Il en est de même pour les fils de travail puisque l'allocation de créneaux à des fils de travail est définie de manière flexible. Au lieu de cela, chaque créneau temporel possède son propre ensemble de registres de contexte pour mémoriser un contexte de travail, qui est utilisé par le fil de travail lorsque le créneau

est alloué au fil de travail, mais pas utilisé lorsque le créneau est alloué au superviseur. Quand un créneau donné est alloué au superviseur, ce créneau utilise à la place le banc de registres de contexte CXS du superviseur. Le superviseur a toujours accès à son propre contexte et aucun des fils de travail ne peut occuper le banc de registres de contexte de superviseur CXS.

[0054] Le fil superviseur SV a la capacité de s'exécuter dans n'importe lequel et dans tous les créneaux temporels S0....S3 (ou plus généralement S0...SJ-1). L'ordonnanceur 24 est agencé de manière à, lorsque le programme dans son ensemble démarre, commencer par allouer le fil superviseur à la totalité des créneaux temporels, c'est-à-dire qu'ainsi le superviseur SV démarre en s'exécutant dans tous les créneaux S0...SJ-1. Toutefois, le fil superviseur est muni d'un mécanisme pour, à un certain point ultérieur (soit immédiatement soit après avoir réalisé une ou plusieurs tâches de superviseur), abandonner temporairement chacun des créneaux dans lequel il s'exécute à l'un respectif des fils de travail, par exemple initialement les fils de travail W0...W3 dans l'exemple représenté en figure 3 (ou plus généralement W0...WJ-1). Cela est obtenu en faisant exécuter une instruction d'exécution par le fil superviseur, qui prend au moins l'adresse d'un fil de travail dans la mémoire d'instructions 12 comme opérande. Les fils de travail sont des portions de code qui peuvent s'exécuter simultanément entre elles, chacune représentant une ou plusieurs tâches de calcul respectives à réaliser.

[0055] L'instruction d'exécution agit sur l'ordonnanceur 24 de manière à abandonner le créneau temporel courant, dans lequel cette instruction est elle-même exécutée, au fil de travail spécifié par l'opérande. On notera qu'il est implicite dans l'instruction d'exécution que c'est le créneau temporel dans lequel cette instruction est exécutée qui est abandonné (implicite dans le contexte d'instructions de code machine signifie qu'il n'y a pas besoin d'opérande pour spécifier cela – c'est sous-entendu de manière implicite d'après le code opération lui-même). Ainsi le créneau temporel qui est abandonné est le créneau temporel dans lequel le superviseur exécute l'instruction d'exécution.

[0056] Le fil superviseur SV réalise une opération similaire dans chacun d'un ou plusieurs autres des créneaux temporels, pour abandonner certains ou la totalité de ses créneaux temporels à des fils respectifs différents parmi les fils de travail W0...WJ-1 (sélectionnés dans un ensemble plus grand de fils de travail possibles dans la mémoire d'instructions 12). Une fois qu'il a réalisé cela pour le dernier créneau, le superviseur est suspendu (puis plus tard il va reprendre là où il avait quitté lorsque l'un des créneaux est rendu par un fil de travail W). Le fil superviseur SV est ainsi capable d'allouer des fils de travail différents, chacun réalisant une ou plusieurs tâches, à différents créneaux des créneaux temporels d'exécution entrelacés S0...SJ-1 (par

exemple J=4 comme cela est illustré, ou J=6). Lorsque le fil superviseur détermine qu'il est temps d'exécuter un fil de travail, il utilise l'instruction d'exécution pour allouer ce fil de travail au créneau temporel dans lequel l'instruction d'exécution a été exécutée.

[0057] Dans certains modes de réalisation, le jeu d'instructions comprend aussi une variante de l'instruction d'exécution, "exécuter tous", ou exécution générale. Cette instruction est utilisée pour lancer un ensemble de plusieurs fils de travail ensemble, tous exécutant le même code. Dans des modes de réalisation cela lance un fil de travail dans chacun des créneaux de l'unité de traitement S0…S3 (ou plus généralement S0…. S(J-1)).

[0058] Une fois lancé, chacun des fils de travail alloués actuellement W0…WJ-1 se déroule pour réaliser lesdites une ou plusieurs tâches de calcul définies dans le code spécifié par l'instruction d'exécution respective. À la fin de cela, le fil de travail respectif rend le créneau temporel dans lequel il s'exécute au fil superviseur. Cela est obtenu en exécutant une instruction de sortie dans le fil de travail respectif. L'instruction de sortie agit sur l'ordonnanceur 24 de façon à repasser le créneau temporel courant, dans lequel cette instruction est elle-même exécutée, au fil superviseur. En réponse l'ordonnanceur 24 continue ensuite à exécuter le superviseur dans ce créneau.

[0059] La figure 4 illustre en outre d'autres exemples du détail du processeur 4, comprenant le détail de l'unité d'exécution 18 et des registres de contexte 26. Le processeur comprend un tampon d'instructions 53 respectif pour chacun des M fils pouvant être exécutés simultanément. Les registres de contexte 26 comprennent un banc de registres principal (MRF) respectif 26M pour chacun des M contextes de travail et pour le contexte de superviseur. Les registres de contexte comprennent en outre un banc de registres auxiliaire (ARF) 26A respectif pour au moins chacun des contextes de travail. Les registres de contexte 26 comprennent en outre un banc de registres de poids commun (WRF) 26W, auquel tous les fils de travail s'exécutant actuellement peuvent accéder en lecture. Le WRF peut être associé au contexte de superviseur en ce que le fil superviseur est le seul fil qui peut écrire dans le WRF. Les registres de contexte 26 peuvent aussi comprendre un groupe respectif de registres d'état de commande 26CSR pour chacun des contextes de superviseur et de travail. L'unité d'exécution 18 comprend une unité d'exécution principale 18M et une unité d'exécution auxiliaire 18A. L'unité d'exécution principale 18M comprend une unité de chargement-stockage (LSU) 55 et une unité logique et arithmétique en entiers (IALU) 56. L'unité d'exécution auxiliaire 18A comprend au moins une unité arithmétique en virgule flottante (FPU).

[0060] Dans chacun des J créneaux temporels entrelacés S0…SJ-1, l'ordonnanceur 24 contrôle l'étage d'extraction 14 pour extraire au moins une instruction d'un fil respectif

à partir de la mémoire d'instructions 11, pour la mettre dans l'un respectif des J tampons d'instructions 53 correspondant au créneau temporel courant. Dans des modes de réalisation, chaque créneau temporel est un cycle d'exécution du processeur, bien que d'autres schémas ne soient pas exclus (par exemple un tour de rôle pondéré). Dans chaque cycle d'exécution du processeur 4 (c'est-à-dire chaque cycle de l'horloge du processeur qui fournit l'horloge au compteur programme) l'étage d'extraction 14 extrait soit une seule instruction soit un petit "groupe d'instructions" (par exemple, un groupe de deux instructions ou un groupe de quatre instructions), en fonction de la mise en œuvre. Chaque instruction est ensuite délivrée, via l'étage de décodage 16, à l'un du LSU 55 ou de l'IALU 56 de l'unité d'exécution principale 18M ou au FPU de l'unité d'exécution auxiliaire 18A, en fonction du fait que l'instruction (selon son code opération) est une instruction d'accès à la mémoire, une instruction arithmétique en entiers ou une instruction arithmétique en virgule flottante, respectivement. Le LSU 55 et l'IALU 56 de l'unité d'exécution principale 18M exécutent leurs instructions en utilisant des registres du MRF 26M, les registres particuliers à l'intérieur du MRF 26M étant spécifiés par des opérandes des instructions. Le FPU de l'unité d'exécution auxiliaire 18A réalise des opérations utilisant des registres se trouvant dans les bancs ARF 26A et WRF 26W, où les registres particuliers dans le banc ARF sont spécifiés par des opérandes des instructions. Dans des modes de réalisation, les registres se trouvant dans le WRF peuvent être implicites dans le type d'instruction (c'est-à-dire prédéterminés pour ce type d'instruction). L'unité d'exécution auxiliaire 18A peut aussi contenir des circuits prenant la forme de bascules logiques internes à l'unité d'exécution auxiliaire 18A pour contenir certains états internes 57 destinés à être utilisés pour réaliser les opérations d'un ou plusieurs des types d'instructions arithmétiques en virgule flottante.

[0061] Dans des modes de réalisation qui extraient et exécutent des instructions en groupes, les instructions individuelles se trouvant dans un groupe d'instructions donné sont exécutées simultanément, en parallèle en suivant des pipelines indépendants 18M, 18A (représentés en figure 4). Dans des modes de réalisation qui exécutent des groupes de deux instructions, les deux instructions peuvent être exécutées simultanément en suivant les pipelines auxiliaire et principal. Dans ce cas, le pipeline principal est agencé pour exécuter des types d'instructions qui utilisent le banc MRF et le pipeline auxiliaire est utilisé pour exécuter des types d'instructions qui utilisent le banc ARF. Le fait d'appairer des instructions en groupes complémentaires appropriés peut être géré par le compilateur.

[0062] Chaque contexte de fil de travail possède sa propre instance du banc de registres principal (MRF) 26M et du banc de registres auxiliaire (ARF) 26A (c'est-à-dire un MRF et un ARF pour chacun des créneaux de fils en barillet). La fonctionnalité décrite

ici en relation avec le MRF ou l'ARF doit être comprise comme fonctionnant sur la base d'un contexte. Toutefois il y a un seul banc de registres de poids partagé (WRF), partagé entre les fils. Chaque fil peut accéder aux bancs MRF et ARF de seulement son propre contexte 26. Toutefois, tous les fils de travail s'exécutant actuellement peuvent accéder au WRF commun. Le WRF fournit ainsi un ensemble commun de poids destiné à être utilisé par tous les fils de travail. Dans des modes de réalisation, seul le superviseur peut écrire dans le WRF, et les fils de travail peuvent seulement lire dans le WRF.

[0063] Le jeu d'instructions du processeur 4 comprend au moins un type d'instruction de chargement dont le code opération, lorsqu'il est exécuté, amène le LSU 55 à charger des données à partir de la mémoire de données 22 dans le banc ARF, 26A, respectif du fil dans lequel les instructions de chargement ont été exécutées. L'emplacement de la destination dans le banc ARF est spécifié par un opérande de l'instruction de chargement. Un autre opérande de l'instruction de chargement spécifie un registre d'adresse dans le MRF, 26M, respectif qui contient un pointeur vers une adresse dans la mémoire de données 22 à partir de laquelle il faut charger les données. Le jeu d'instructions du processeur 4 comprend aussi au moins un type d'instruction de stockage dont le code opération, lorsqu'il est exécuté, amène le LSU 55 à stocker des données dans la mémoire de données 22 à partir du banc ARF respectif du fil dans lequel l'instruction de stockage a été exécutée. L'emplacement de la source du stockage dans le banc ARF est spécifié par un opérande de l'instruction de chargement. Un autre opérande de l'instruction de chargement spécifie un registre d'adresse dans le MRF, qui contient un pointeur vers une adresse dans la mémoire de données 22, à laquelle il faut stocker la donnée. En général, le jeu d'instructions peut comprendre des types d'instructions de chargement et de stockage séparés, et/ou au moins un type d'instruction de chargement-stockage qui combine les opérations de chargement et de stockage dans une seule instruction. Comme on va le décrire plus en détail dans la suite, le jeu d'instructions peut inclure un type particulier d'instruction de chargement-stockage qui réalise deux chargements et une opération de stockage, le tout en une seule instruction. On notera que le jeu d'instructions d'un processeur 4 donné peut inclure de multiples variétés différentes de types d'instructions de chargement, de stockage et/ou de chargement-stockage.

[0064] Le jeu d'instructions du processeur comprend aussi un ou plusieurs types d'instructions arithmétiques pour réaliser des opérations arithmétiques. Selon des modes de réalisation décrits ici, cela peut inclure au moins un type d'instruction arithmétique qui fait usage du banc de registres de poids commun, WRF, 26W. Ce type d'instruction prend au moins un opérande qui spécifie au moins une source de l'opération arithmétique correspondante dans le banc ARF, 26A, respectif du fil dans

lequel l'instruction arithmétique a été exécutée. Toutefois, au moins une autre source de l'instruction arithmétique est le WRF commun, commun à tous les fils de travail. Dans des modes de réalisation, cette source est implicite dans l'instruction arithmétique en question (c'est-à-dire implicite pour ce type d'instruction arithmétique). Implicite dans le sens d'une instruction de code machine signifie qu'il n'y a pas besoin de spécifier un opérande. C'est-à-dire que dans ce cas, l'emplacement de la source dans le WRF est inhérent d'après le code opération (prédéterminé pour ce code opération particulier). En variante dans d'autres modes de réalisation, l'instruction arithmétique peut prendre un opérande spécifiant dans quel ensemble de registres de poids on doit prendre les poids, parmi quelques ensembles différents se trouvant dans le WRF. Toutefois, le fait que la source des poids se trouve dans le banc WRF (par opposition disons au banc MRF ou au banc ARF d'usage général) est encore implicite.

[0065] En réponse au code opération du type concerné d'instruction arithmétique, l'unité arithmétique (par exemple le FPU) dans l'unité d'exécution auxiliaire 18A réalise une opération arithmétique, comme spécifié par le code opération, qui comprend d'opérer sur les valeurs se trouvant dans le ou les registres de source spécifiés dans le banc ARF respectif du fil, et dans le ou les registres de source dans le WRF. Elle fournit aussi en sortie un résultat de l'opération arithmétique à un registre de destination dans le banc ARF respectif du fil comme spécifié explicitement par un opérande de destination de l'instruction arithmétique.

[0066] Des exemples de types d'instructions arithmétiques qui peuvent utiliser une source dans le WRF commun, 26W, peuvent inclure : un ou plusieurs types d'instructions de multiplication vectorielle, un ou plusieurs types d'instructions de multiplication matricielle, un ou plusieurs types d'instructions de multiplication vectorielle à accumulation et/ou types d'instructions de multiplication matricielle à accumulation (qui accumulent un résultat de la multiplication entre une instance donnée de l'instruction et la suivante), et/ou un ou plusieurs types d'instructions de convolution. Par exemple, un type d'instruction de multiplication vectorielle peut multiplier un vecteur d'entrée explicite provenant du banc ARF, 26A, par un vecteur prédéterminé de poids provenant du WRF ; ou un type d'instruction de multiplication matricielle peut multiplier un vecteur d'entrée explicite provenant du banc ARF par une matrice prédéterminée de poids provenant du WRF. Dans un autre exemple, un type d'instruction de convolution peut faire une convolution d'une matrice d'entrée provenant du banc ARF avec une matrice prédéterminée provenant du WRF. Le fait d'avoir un banc de registres de poids partagés, WRF, commun à une pluralité de fils, permet à chaque fil de multiplier ou faire une convolution entre un noyau commun et ses propres données respectives. Cela est utile puisque c'est d'un scénario qui survient souvent dans

beaucoup d'applications d'apprentissage automatique, par exemple là où chaque fil représente un nœud différent dans un réseau neuronal et où le noyau commun représente une caractéristique qui est recherchée ou pour laquelle on réalise un apprentissage (par exemple une arête ou une forme particulière dans une région ou un volume de données graphiques).

[0067] Dans des modes de réalisation, les valeurs se trouvant dans le WRF, 26W, peuvent être écrites par le fil superviseur. Le superviseur (qui dans un mode de réalisation commence par s'exécuter dans tous les créneaux S0...SM) exécute en premier une série d'instructions de mise en place pour écrire les valeurs de certains poids commun dans l'emplacement prédéterminé dans le WRF. Il exécute ensuite des instructions d'exécution (ou une instruction d'exécution générale) pour lancer un fil de travail respectif dans certains ou la totalité des créneaux S0...SJ-1. Chaque fil de travail comprend ensuite une ou plusieurs instances d'une ou plusieurs instructions arithmétiques du ou des types décrits précédemment, de manière à réaliser la ou les opérations arithmétiques correspondantes sur ses propres données d'entrée respectives, chargées dans son banc ARF, 26A, respectif, mais en utilisant les poids communs écrits par le superviseur dans le WRF, 26W. Lorsque chaque fil termine sa tâche respective, il exécute une instruction de sortie pour rendre son créneau au superviseur. Lorsque tous les fils lancés ont terminé leurs tâches respectives, le superviseur peut écrire de nouvelles valeurs dans le WRF et lancer un nouvel ensemble de fils (ou lancer un nouvel ensemble pour continuer à utiliser les valeurs existantes dans le WRF).

[0068] On notera que les appellations "principal", "auxiliaire" et "poids" ne sont pas nécessairement limitatives. Dans des modes de réalisation il peut y avoir un premier banc de registres (par contexte de travail), un deuxième banc de registres (par contexte de travail) et un troisième banc de registres partagé (par exemple faisant partie du contexte de superviseur mais accessible à tous les fils de travail). Le banc ARF 26A et l'unité d'exécution auxiliaire 18 peuvent aussi être appelés banc de registres arithmétiques et unité d'exécution arithmétique puisqu'ils sont utilisés pour des instructions arithmétiques (ou au moins l'arithmétique en virgule flottante). Le MRF 26M et l'unité d'exécution auxiliaire 18 peuvent aussi être appelés banc de registres d'adresses mémoire et unité d'exécution arithmétique puisque l'une de leurs utilisations consiste à accéder à la mémoire. Le banc de registres de poids (WRF) 26W est appelé ainsi car il est utilisé pour contenir des poids multiplicateurs utilisés dans un certain type ou dans certains types d'instructions arithmétiques, qui seront décrits bientôt dans la suite. Par exemple cela pourrait être utilisé pour représenter les poids de nœuds dans un réseau neuronal. Dit d'une autre manière, le banc MRF pourrait être appelé banc de registres d'entiers puisqu'il est utilisé pour contenir des opérandes entiers, tandis que le banc

ARF pourrait être appelé banc de registres de virgule flottante puisqu'il est utilisé pour contenir des opérandes en virgule flottante. Dans des modes de réalisation qui exécutent des instructions en groupes de deux, le MRF est le banc de registre utilisé par le pipeline principal et le banc ARF est le registre utilisé par le pipeline auxiliaire.

[0069] Cependant, dans des variantes de réalisation, on notera que l'espace de registres 26 n'est pas nécessairement divisé en ces bancs de registres séparés pour ces différents buts. Au lieu de cela, des instructions exécutées dans les unités d'exécution principale et auxiliaire peuvent être capables de spécifier des registres dans le même banc de registres partagé (un banc de registres par contexte dans le cas d'un processeur à fils d'exécution multiples). Aussi, le pipeline 13 ne comporte pas nécessairement des pipelines constituants parallèles (par exemple des pipelines auxiliaire et principal) pour exécuter simultanément des groupes d'instructions.

[0070] Le processeur 4 peut aussi comprendre une interface d'échange 51 pour échanger des données entre la mémoire 11 et une ou plusieurs autres ressources, par exemple d'autres instances de processeurs et/ou des dispositifs externes comme une interface réseau ou un dispositif de stockage attaché à un réseau (NAS). Comme cela est illustré en figure 5, dans des modes de réalisation le processeur 4 peut constituer l'un des pavés d'une matrice 6 de pavés processeurs interconnectés, chaque pavé exécutant une partie d'un programme plus large. Les processeurs individuels 4 (pavés) constituent ainsi une partie d'un processeur ou d'un système de traitement plus large 6. Les pavés 4 peuvent être interconnectés entre eux via un sous-système d'interconnexion 34, auquel ils sont connectés via leur interface d'échange 51 respective. Les pavés 4 peuvent être mis en œuvre sur la même puce (c'est-à-dire la même puce élémentaire) ou sur des puces différentes, ou dans une combinaison (c'est-à-dire que la matrice peut être constituée de multiples puces comprenant chacune de multiples pavés 4). Le système d'interconnexion 34 et l'interface d'échange 51 peuvent par conséquent comprendre un mécanisme d'interconnexion interne (sur la puce) et/ou un mécanisme d'échange externe (entre puces), en conséquence.

[0071] Dans un exemple d'application d'un processeur ou système à fils d'exécution multiples et/ou à pavés multiples, le programme exécuté à travers les multiples fils et/ou pavés 4 comprend un algorithme d'intelligence artificielle, comme un algorithme agencé pour réaliser l'apprentissage d'un réseau neuronal et/ou pour réaliser des inférences sur la base d'un réseau neuronal. Dans de tels modes de réalisation chaque fil de travail, ou la partie du programme exécutée sur chaque pavé, ou chaque fil de travail sur chaque pavé, est utilisé pour représenter un nœud 102 différent dans un réseau neuronal (un type de graphe) ; et des communications entre fils et/ou pavés, en conséquence, représentent des arêtes 104 entre des nœuds 102 dans le graphe. Cela est illustré en figure 6.

[0072] L'intelligence artificielle commence par une étape d'apprentissage dans laquelle l'algorithme d'intelligence artificielle apprend un modèle de connaissance. Le modèle comprend un graphe de nœuds interconnectés (c'est-à-dire des sommets) 102 et d'arêtes (c'est-à-dire des liaisons) 104. Chaque nœud 102 du graphe comporte une ou plusieurs arêtes d'entrée et une ou plusieurs arêtes de sortie. Certaines des arêtes d'entrée de certains des nœuds 102 sont les arêtes de sortie de certains autres nœuds, connectant ainsi entre eux les nœuds pour former le graphe. En outre, une ou plusieurs des arêtes d'entrée d'un ou plusieurs des nœuds 102 constituent les entrées du graphe dans son ensemble, et une ou plusieurs des arêtes de sortie d'un ou plusieurs des nœuds 102 constituent les sorties du graphe dans son ensemble. Parfois, un nœud donné peut même comporter tous ces éléments : des entrées du graphe, des sorties du graphe et des connexions à d'autres nœuds. Chaque arrête 104 communique une valeur ou plus souvent un tenseur (une matrice à n dimensions), cela constituant les entrées et les sorties fournies aux nœuds et reçues des nœuds 102 sur leur arrêtes d'entrée et de sortie respectivement.

[0073] Chaque nœud 102 représente une fonction de son ou de ses entrées reçues sur son arrête ou ses arrêtes d'entrée, le résultat de cette fonction étant la ou les sorties fournies sur l'arrête ou les arrêtes de sortie. Chaque fonction est paramétrée par un ou plusieurs paramètres respectifs (parfois appelés poids, bien qu'ils n'aient pas besoin nécessairement d'être des poids multiplicateurs). En général, les fonctions représentées par les différents nœuds 102 peuvent prendre différentes formes de fonctions et/ou peuvent être paramétrées par différents paramètres.

[0074] En outre, chacun desdits un ou plusieurs paramètres de chaque fonction d'un nœud est caractérisé par une valeur d'erreur respective. En outre, une condition respective peut être associée à l'erreur ou aux erreurs dans le ou les paramètres de chaque nœud 102. Pour un nœud 102 représentant une fonction paramétrée par un seul paramètre, la condition peut être un simple seuil, c'est-à-dire que la condition est satisfaite si l'erreur se trouve dans les limites du seuil spécifié mais n'est pas satisfaite si l'erreur est au-delà du seuil. Pour un nœud 102 paramétré par plus qu'un seul paramètre respectif, la condition pour que ce nœud 102 ait atteint un niveau d'erreur acceptable peut être plus complexe. Par exemple, la condition peut être satisfaite seulement si chacun des paramètres de ce nœud 102 se trouve dans les limites du seuil respectif. Dans un autre exemple, une métrique combinée peut être définie comme combinant les erreurs dans les différents paramètres pour le même nœud 102, et la condition peut être satisfaite à condition que la valeur de la métrique combinée se trouve dans les limites d'un seuil spécifié, mais autrement la condition n'est pas satisfaite si la valeur de la métrique combinée est au-delà du seuil (ou vice versa en fonction de la définition de la métrique). Quelle que soit la condition, cela donne une mesure indiquant si l'erreur

dans le ou les paramètres du nœud descend en dessous d'un certain niveau ou degré d'acceptabilité. En général on peut utiliser toute métrique appropriée. La condition ou la métrique peut être la même pour tous les nœuds, ou peut être différente pour certains nœuds respectifs différents parmi les nœuds.

[0075] Dans l'étape d'apprentissage l'algorithme reçoit des données expérimentales, c'est-à-dire de multiples points de données représentant différentes combinaisons possibles d'entrées dans le graphe. Au fur et à mesure que des données expérimentales sont reçues, l'algorithme ajuste progressivement les paramètres des divers nœuds 102 du graphe sur la base des données expérimentales de manière à essayer de minimiser les erreurs dans les paramètres. Le but est de trouver des valeurs des paramètres telles que la sortie du graphe soit aussi proche que possible d'une sortie souhaitée pour une entrée donnée. Lorsque le graphe dans son ensemble tend vers un tel état, on dit que le graphe converge. Après l'obtention d'un degré de convergence approprié, le graphe peut être utilisé pour réaliser des prédictions ou des inférences, c'est-à-dire pour prédire un résultat pour certaines entrées données ou pour inférer une cause pour une sortie donnée.

[0076] L'étape d'apprentissage peut prendre un certain nombre de formes possibles différentes. Par exemple, dans une approche supervisée, les données expérimentales d'entrée prennent la forme de données d'apprentissage, c'est-à-dire d'entrées qui correspondent à des sorties connues. Avec chaque point de données, l'algorithme peut ajuster les paramètres de telle sorte que la sortie concorde le plus précisément possible avec la sortie connue pour l'entrée donnée. Dans l'étape de prédiction ultérieure, le graphe peut alors être utilisé pour mapper une interrogation d'entrée sur une sortie prédite approximative (ou vice versa si on réalise une inférence). D'autres approches sont aussi possibles. Par exemple, dans une approche non supervisée, il n'y a pas de concept de résultat de référence pour chaque donnée d'entrée, et au lieu de cela on laisse l'algorithme d'intelligence artificielle identifier sa propre structure dans les données de sortie. Ou dans une approche par renforcement, l'algorithme essaye au moins une sortie possible pour chaque point de données dans les données expérimentales, et on lui indique si sa sortie est positive ou négative (et potentiellement un degré avec lequel elle est positive ou négative), par exemple gagné ou perdu, ou récompense ou cout, ou similaire. Sur de nombreux essais, l'algorithme peut progressivement ajuster les paramètres du graphe pour être capable de prédire des entrées qui vont conduire à un résultat positif. Les diverses approches et divers algorithmes pour l'apprentissage d'un graphe sont connus de l'homme de l'art de l'apprentissage automatique.

[0077] Selon un exemple d'application des techniques décrites ici, chaque fil du travail est programmé pour réaliser les calculs associés à un nœud individuel respectif des nœuds

102 dans un graphe d'intelligence artificielle comme un réseau neuronal. Dans ce cas, au moins certaines des arêtes 104 entre les nœuds 102 correspondent aux échanges de données entre des fils, et certaines peuvent impliquer des échanges entre des pavés. Dans le cas d'un agencement multi-pavé 6 avec plusieurs fils par pavé 4, chaque pavé 4 exécute un sous-graphe du graphe. Chaque sous-graphe comprend un sous-programme superviseur comprenant un ou plusieurs fils superviseurs, et un ensemble de fils de travail représentant les nœuds 102 du sous-graphe respectif.

[0078] Dans des applications comme l'intelligence artificielle, il serait souhaitable de pouvoir acheminer un flux de données de manière efficace vers et à partir de la mémoire de données 22 du processeur 4 (par exemple un pavé). Par exemple cela serait particulièrement utile (mais pas exclusivement) pour accompagner une séquence d'instructions arithmétiques complexes comme des instructions de produits scalaires vectoriels, de produits matriciels, de produits scalaires vectoriels à accumulation, de produits matriciels à accumulation ou des instructions de convolution dédiées, qui englobent un fort degré de complexité arithmétique dans une seule instruction arithmétique.

[0079] Pour répondre à ces problèmes, des modes de réalisation prévoient un type d'instruction de chargement-stockage, dans le jeu d'instructions du processeur 4, qui réalise deux opérations de chargement et une opération de stockage, puis applique ensuite automatiquement un pas de progression indépendant à chacune des adresses de chargement et de stockage, tout cela en réponse à une seule instance du code opération de l'instruction. En outre, l'instruction utilise un empaquetage efficace d'adresses dans le banc de registres approprié (par exemple le banc MRF 26M), d'où il résulte que les trois adresses (les deux adresses de chargement et l'adresse de stockage) sont empaquetées dans l'espace de deux registres. Cela permet aux trois adresses mémoire et au registre de pas d'être accédés en une seule instruction avec seulement trois ports d'accès à partir du MRF 26M, vers le LSU (un port respectif pour accéder à chacun des trois registres).

[0080] L'instruction peut être appelée ici "ld2xst64pace", en faisant référence au fait qu'elle charge deux valeurs de 64 bits et stocke une valeur de 64 bits, le terme "pace" étant synonyme de pas de progression. Toutefois cette étiquette ou les largeurs de bits particulières ne doivent pas être considérées comme étant nécessairement limitatives. L'instruction peut de manière plus générale être appelée simplement instruction de chargement-stockage (bien que cela n'implique pas qu'elle soit nécessairement la seule sorte d'instructions de chargement-stockage dans le jeu d'instructions du processeur), ou "ldx2st" (chargement fois deux et stockage). Sa syntaxe est la suivante :

[0081] ld2xst $aDst, $aSrc, $mAddr, $mStride, Strimm

[0082] $aDst fait référence à un ou plusieurs opérandes identifiant une destination dans le

banc de registres auxiliaire (ou arithmétique) (ARF) 26A, la destination comprenant un ou plusieurs des registres se trouvant dans le banc ARF. Dans des modes de réalisation, chacun des registres se trouvant dans le banc ARF a une largeur de 32 bits, et la destination $aDst peut être constituée de quatre registres de 32 bits de la sorte : $aDst0:Dst0+3. L'opérande de destination du ld2xst64pace peut spécifier l'emplacement de seulement l'un de ces registres de destination dans le banc ARF (par exemple le plus bas $aDst0) et les emplacements de l'autre ou des autres peuvent être implicites par rapport à celui-ci (par exemple, les trois registres contigus suivants dans le banc ARF). En variante dans d'autres mises en œuvre, il n'est pas exclu que chacun des multiples opérandes de destination puisse être identifié explicitement ou indépendamment par des opérandes séparés (bien que cela nécessite une largeur d'instruction augmentée).

[0083] $aSrc fait référence à un ou plusieurs opérandes identifiant une source dans le banc ARF, 26A, la source comprenant un ou plusieurs des registres se trouvant dans le banc ARF. Dans des modes de réalisation chacun des registres se trouvant dans le banc ARF a une largeur de 32 bits, et la source $aSrc peut être constituée de deux registres de 32 bits de la sorte : $aSrc0:Src+1. L'opérande de source du ld2xst64pace peut spécifier l'emplacement de seulement l'un de ces trois registres de source dans le banc ARF (par exemple le plus bas $aSrc0) et l'emplacement de l'autre ou des autres peut être implicite par rapport à celui-ci (par exemple le registre contigu suivant dans le banc ARF). En variante dans d'autres mise en œuvre, il n'est pas exclu que chacun des multiples opérandes de source puisse être identifié explicitement et indépendamment par des opérandes séparés (bien que cela nécessite une largeur d'instruction augmentée).

[0084] $mAddr fait référence à un ou plusieurs opérandes spécifiant l'emplacement de deux registres $mAddr0:Addr0+1 dans le banc de registres principal (MRF) 25M, qui entre eux contiennent trois adresses : deux adresses de chargement et une adresse de stockage. Dans des modes de réalisation chacun des registres se trouvant dans le MRF a une largeur de 32 bits. L'opérande d'adresse mémoire peut spécifier l'emplacement de seulement l'un de ces registres dans le MRF (par exemple le plus bas $mAddr0), l'emplacement de l'autre étant implicite par rapport à celui-ci (par exemple le registre contigu suivant dans le MRF). En variante, dans d'autres mises en œuvre, il n'est pas exclu que chacun des registres d'adresses mémoire puisse être identifié explicitement et indépendamment par des opérandes séparés (bien que cela nécessite une largeur d'instruction augmentée).

[0085] Strimm est un ensemble d'opérandes immédiats, un opérande respectif pour chacune des adresses des deux chargements et du stockage. Dans le MRF, 26M, on prévoit un registre de pas qui comprend une pluralité de champs. Chaque champ contient une

valeur respective différente d'un ensemble prédéterminé de valeurs de pas possibles différentes. Une valeur de pas est une valeur de laquelle on incrémente une adresse mémoire, c'est-à-dire un pas d'adresse mémoire, typiquement destiné à une utilisation dans une série de tels pas. Pour chacun des deux chargements et du stockage, l'opérande de pas immédiat respectif spécifie à partir de quel champ dans le registre de pas il faut prendre la valeur de pas à appliquer à l'adresse respective de chargement ou de stockage, après avoir réalisé l'opération de chargement ou de stockage respective de la présente instance de l'instruction. Cela réalise un déplacement le long de l'adresse dans le MRF, 26M, au bénéfice d'une instance suivante de l'instruction de chargement-stockage.

[0086] Dans des modes de réalisation, l'unité de chargement-stockage 55 supporte une fonctionnalité par laquelle une valeur spéciale de l'opérande de pas immédiat peut directement spécifier une valeur de pas de 1 (c'est-à-dire une augmentation d'une unité d'adresse dans l'espace d'adresses utilisé), plutôt que de pointer vers un champ de registre. C'est-à-dire qu'en dehors de la plage de valeurs que l'opérande de pas immédiat peut prendre, l'une des valeurs spécifie un pas de un, et certaines ou la totalité des autres spécifient des champs possibles différents dans le registre de pas contenant des valeurs de pas programmables. Une unité désigne ici la taille atomique de l'accès aux données. Par exemple, si l'opérande est un vecteur à 4 éléments de valeurs en virgule flottante de 16 bits, l'incrément est de 1 atome/unité, ce qui est équivalent à 8 multiplets (64 bits).

[0087] Le registre de pas est un registre dans le banc MRF, 26M. Dans des modes de réalisation le fil de travail lui-même est responsable de l'écriture des valeurs de pas dans les champs de son propre registre de pas (dans son propre banc MRF). En variante, il n'est pas exclu que dans d'autres mises en œuvre les valeurs de pas se trouvant dans les champs du registre de pas puissent être écrites par le fil superviseur SV, ou bien qu'une combinaison d'approches soit utilisée.

[0088] L'obligation d'avoir un opérande spécifiant $mStride est une fonctionnalité optionnelle du processeur 4 dépendant de la mise en œuvre. Dans des modes de réalisation, l'instruction de chargement-stockage prend un opérande spécifiant l'emplacement du registre de pas $mStride dans le MRF, 26M. Ainsi le programme peut sélectionner le registre de pas parmi une pluralité de registres de pas possibles, assurant encore davantage de flexibilité dans la sélection du pas. Toutefois, dans des variantes de réalisation il n'est pas exclu que l'emplacement de $mStride puissent être fixe ou implicite et ne nécessite pas d'opérande dans l'instruction de chargement-stockage.

[0089] La figure 7 illustre un exemple de l'empaquetage de trois adresses dans deux registres de 32 bits dans le MRF, 26M, chaque adresse étant sur 21 bits. La figure 7

illustre aussi un exemple de l'empaquetage de trois valeurs de pas dans un registre de 32 bits dans le MRF, chaque valeur de pas étant sur 10 bits (dans ce cas deux bits du registre restent inutilisés). Ainsi l'instruction de chargement-stockage peut accéder à trois adresses mémoire via seulement deux ports larges de 32 bits du MRF en une seule instruction. Un troisième port large de 32 bits peut être utilisé par la même instruction pour accéder au registre de pas, contenant par exemple trois valeurs de pas de 10 bits (deltas d'adresse) dans trois champs larges de 10 bits disposés dans un registre de pas large de 32 bits dans le MRF (laissant 2 bits de ce registre inutilisés). On notera que la disposition particulière représentée en figure 7 n'est qu'un exemple. Par exemple, dans une autre mise en œuvre addr2 peut être à cheval sur les deux registres, plutôt que addr1.

[0090] La figure 8 illustre un agencement de multiples registres de pas $mStrideA, $mStrideB, $mStrideC … dans le MRF, 26M. L'un quelconque de ces registres pourrait être spécifié comme étant le registre de pas $mStride par l'opérande correspondant de l'instruction de chargement-stockage. Chaque registre de pas possible comprend une pluralité de champs, par exemple dans des modes de réalisation au nombre de trois (bien que cela dépende de la mise en œuvre il n'y a pas nécessairement le même nombre de champs ni de registres de pas qu'il n'y a d'adresses de chargement et de stockage). Pour chacune des adresses des deux chargements et du stockage, l'opérande immédiat respectif dans l'ensemble de valeurs immédiates Strimm peut spécifier l'un des champs possibles dans le registre de pas $mStride à partir duquel il faut prendre une valeur de pas respective (un delta d'adresse à appliquer à la suite du chargement ou du stockage respectif). Par exemple, dans des modes de réalisation, si l'opérande de pas est de 2 bits pour chacun des deux chargements et du stockage, alors trois des valeurs possibles spécifient des champs différents parmi les trois champs de pas, et l'autre valeur possible spécifie simplement un pas de un sans faire référence à une valeur contenue dans un registre. Par exemple 00 va spécifier un pas de 1 (atome), tandis que le 01 va spécifier un premier champ dans le registre de pas, 10 va spécifier un deuxième champ dans le registre de pas et 11 va spécifier un troisième champ dans le registre de pas.

[0091] On notera que dans des modes de réalisation, l'une des valeurs possibles de la valeur immédiate de pas spécifie un incrément par défaut de 1, au lieu d'un champ dans le registre de pas $mStride.

[0092] En fonctionnement, lorsqu'il est exécuté par l'unité d'exécution 18, le code opération de l'instruction de chargement-stockage (après décodage par l'étage de décodage 16) déclenche le LSU 55 pour réaliser les opérations suivantes. Il charge les valeurs à partir des deux adresses de chargement dans la mémoire 22, comme spécifié par l'adresse contenue dans $mAddr dans le MRF, 26M, dans la destination dans le banc

ARF, 26A, spécifié par $aDsc. En outre, il mémorise la ou les valeurs provenant de $aSrc dans le banc ARF dans l'adresse de stockage en mémoire 22 spécifiée par l'adresse contenue dans $mAddr dans le banc MRF. Le LSU 55 post-incrémente ensuite indépendamment chacune des deux adresses de chargement et l'adresse de stockage de la valeur de pas respective provenant du champ respectif du registre de pas $mStride dans le MRF, spécifié par l'opérande respectif parmi les trois opérandes immédiats Strimm.

[0093] On notera que pour chacune des adresses des deux chargements et du stockage de l'instance courante de l'instruction de chargement-stockage, l'opération de pas de progression de l'instruction courante incrémente l'adresse respective du pas respectif à la suite de chacune des opérations de chargement et de stockage de l'opération de chargement-stockage courante, respectivement. En fonction de la mise en œuvre, cela peut signifier que tous les incréments sont appliqués ensemble après les deux chargements et le stockage, par exemple, chargement, chargement, stockage, incrément, incrément, incrément. En variante, l'opération de pas peut incrémenter l'adresse respective immédiatement *à la suite* de chaque chargement et du stockage, par exemple chargement, incrément, chargement, incrément, stockage, incrément. En effet le stockage pourrait aussi venir avant l'un des chargements ou avant les deux. Ce qui importe c'est simplement que l'incrément de chaque adresse de chargement soit fait après son chargement respectif, et l'incrément de l'adresse de stockage soit fait après le stockage. L'essentiel est de se déplacer suivant les adresses de chargement et de stockage dans le MRF pour être prêt pour une instance suivante de l'instruction de chargement-stockage.

[0094] Dans des modes de réalisation, la mémoire 11 (ou au moins la mémoire de données 22) comporte seulement deux ports d'accès de 64 bits pour charger des données à partir de la mémoire, et seulement un port d'accès d'une largeur de 64 bits pour stocker des données dans la mémoire. Dans des modes de réalisation, le MRF, 26M (d'un contexte donné) comporte seulement trois ports d'une largeur de 32 bits vers l'unité de chargement-stockage 55 ; et le banc ARF, 26A (d'un contexte donné) comporte seulement un port d'une largeur de 64 bits vers l'unité de chargement-stockage 55. (On notera que dans l'exemple illustré, l'IALU 56 est utilisé pour récupérer les pointeurs dans le MRF 26M et pour calculer les adresses à partir de ceux-ci afin de les passer au LSU 55, et ainsi en effet l'IALU 56 agit comme partie du LSU 55. Par conséquent dans cet exemple les trois ports allant du MRF 26M vers le LSU 55 comprennent les trois ports allant du MRF 26M vers l'IALU 56. Aussi il n'est pas exclu que dans d'autres mises en œuvre le LSU 55 puisse récupérer les pointeurs dans le MRF 26M directement et calculer ses propres adresses sur la base de ceux-ci).

[0095] Dans des modes de réalisation, les quatre registres de destination de 32 bits

$aDst0:Dst0+3 dans le banc ARF (128 bits au total) peuvent être utilisés pour charger, à partir de la mémoire 22, par exemple un vecteur de quatre éléments de valeurs en virgule flottante de 16 bits (f16) et un vecteur de deux éléments de valeurs en virgule flottante de 32 bits (f32). Les deux registres de source de 32 bits $aSrc0:Src+1 se trouvant dans le banc ARF (64 bits au total) peuvent être utilisés pour mémoriser, dans la mémoire 22, par exemple un vecteur de deux éléments de 32 bits de valeurs f32.

[0096] Pour être utilisée dans un programme, l'instruction de chargement-stockage est intercalée parmi d'autres types d'instructions comme des instructions arithmétiques qui prennent des entrées à partir des destinations des instructions de chargement, réalisent des opérations sur la base de celles-ci, et fournissent des résultats aux sources de l'instruction de chargement-stockage. C'est-à-dire que le programme comprend des instances de l'instruction de chargement-stockage et des instances d'au moins une instruction arithmétique, comme l'instruction de produit scalaire vectoriel, l'instruction de produit matriciel, l'instruction de produit scalaire vectoriel à accumulation, l'instruction de produit matriciel à accumulation ou l'instruction de convolution mentionnées précédemment ; les destinations d'au moins certaines des instructions de chargement-stockage étant les sources d'au moins certaines des instructions arithmétiques, et les destinations d'au moins certaines des instructions arithmétiques étant les sources d'au moins certaines des instructions de chargement-stockage. Grâce à la forte densité sémantique de l'instruction de chargement-stockage, avec ses deux chargements et sa fonctionnalité de pas, cela permet au programme de travailler efficacement à travers les données avec une faible charge de code dépensée pour les opérations de chargement et de stockage.

[0097] Par exemple, le programme peut comprendre une série de paires d'instructions, chaque paire d'instructions étant constituée d'une instance de l'instruction de chargement-stockage, suivie d'une instance correspondante d'une instruction arithmétique. Dans chaque paire d'instructions, la source de l'instruction de chargement-stockage est définie comme étant la destination de l'instruction arithmétique provenant d'une paire précédente, et les destinations de l'instruction de chargement-stockage sont définies comme étant les sources de l'instruction arithmétique correspondante dans la paire courante ou une paire ultérieure. Par exemple, considérons une instruction arithmétique "arith" ayant comme syntaxe :

[0098] arith $aDst, $aSrcA, $aSrcB

[0099] où $aDst désigne un opérande spécifiant une destination dans le banc ARF, 26A ; et $aSrcA, $aSrcB désignent des opérandes spécifiant deux sources dans le banc ARF (on notera que "arith" est un nom générique utilisé ici pour toute instruction arithmétique ayant au moins cette syntaxe). Le programme peut ensuite être programmé avec une série de paires d'instructions, par exemple :

[0100] …

[0101] ldx2st Xin-Pin, Pout, Tripacked, Strides; arith Pout, Xin, Pin ;

[0102] ldx2st Xin-Pin, Pout, Tripacked, Strides; arith Pout, Xin, Pin ;

[0103] ldx2st Xin-Pin, Pout, Tripacked, Strides; arith Pout, Xin, Pin ;

[0104] ldx2st Xin-Pin, Pout, Tripacked, Strides; arith Pout, Xin, Pin ;

[0105] …

[0106] ou :

[0107] …

[0108] ldx2st Xin-Pin, Pout_A, Tripacked, Strides; arith Pout_A, Xin, Pin ;

[0109] ldx2st Xin-Pin, Pout_B, Tripacked, Strides; arith Pout_B, Xin, Pin ;

[0110] ldx2st Xin-Pin, Pout_A, Tripacked, Strides; arith Pout_A, Xin, Pin ;

[0111] ldx2st Xin-Pin, Pout_B, Tripacked, Strides; arith Pout_B, Xin, Pin ;

[0112] …

[0113] etc.

[0114] Dans des modes de réalisation, chaque paire est un groupe d'instructions, c'est-à-dire des instructions exécutées simultanément en suivant des pipelines respectifs. Par exemple, dans des modes de réalisation l'instruction de chargement-stockage est exécutée par le pipeline principal en utilisant le banc MRF, 26M, et l'instruction arithmétique est exécutée en parallèle par le pipeline auxiliaire en utilisant le banc ARF, 26A. Toutefois, dans des variantes de réalisation il n'est pas exclu que l'instruction de chargement-stockage et l'instruction arithmétique puissent être exécutées l'une après l'autre par l'intermédiaire d'une seule unité de traitement en pipeline.

[0115] Dans les instances de l'instruction de chargement et de stockage (ldx2st), Xin-Pin désigne les destinations $aDsc des deux opérations de chargement dans le banc ARF, 26A (par exemple un vecteur d'entrée 4xf16 Xin et un vecteur d'entrée 2xf32 Pin chargés dans quatre registres de 32 bits $aDsc0:Dsc0+3) ; et Pout_A ou Pout_B désigne la source du stockage dans le banc ARF (par exemple 2 valeurs f32 mémorisées à partir de deux registres de 32 bits $aSrc0:Src0+1). "Tripacked" désigne la paire de registres d'adresses en paquets de trois $mAddr0:Addr0+1 dans le banc MRF, 26M (par exemple contenant trois pointeurs d'adresse de 21 bits dans deux registres de 32 bits. "Strides" désigne les opérandes de pas $mStride et Strimm, qui spécifient le pas pour chacun des deux chargements et pour l'opération de stockage, en faisant référence à un champ dans le registre de pas $mStride dans le MRF spécifié par trois valeurs immédiates Strimm (voir de nouveau la figure 8). Dans les instances de l'instruction arithmétique, Xin, Pin désignent les registres de source à partir desquels sont prises les entrées de l'instruction arithmétique, qui sont définis comme étant les mêmes que les destinations du chargement réalisé par l'instruction de chargement-

stockage dans la paire d'instructions courante ou une paire d'instructions précédente. Pout désigne la destination pour la sortie de l'instruction arithmétique dans le banc ARF, qui est définie comme étant la même que la source de l'opération de stockage de l'instruction de chargement-stockage dans une paire d'instructions ultérieure. Dans des modes de réalisation qui exécutent chaque paire sous forme d'un groupe de deux instructions simultanées, Xin, Pin sont définis comme étant les mêmes que les destinations du chargement réalisé par l'instruction de chargement-stockage dans le groupe précédent ; et Pout est défini comme étant identique à la source de l'opération de stockage du ldx2st dans le groupe suivant. Cela réutilise les mêmes registres, mais puisque les deux instructions sont exécutées en parallèle, la donnée d'entrée pour l'instruction arithmétique courante est celle qui a été lue par l'instruction de chargement précédente. Le ldx2st courant réutilise ensuite ces mêmes registres pour préparer les données d'entrée pour l'instruction arithmétique suivante.

[0116] Ainsi, dans chaque paire, le chargement-stockage (ldx2st) mémorise la sortie courante trouvée dans un registre Pout provenant d'une instruction arithmétique précédente dans une paire précédente d'instructions précédentes (par exemple un groupe), et aussi charge des valeurs à partir de la mémoire dans des registres Xin et Pin. L'instruction arithmétique suivante dans la même paire (par exemple un groupe) réalise une opération arithmétique sur la base de valeurs chargées précédemment, et fournit sa sortie à un registre Pout pour mémorisation par l'opération de stockage d'une instruction de chargement-stockage ultérieure dans une paire ultérieure.

[0117] On notera que dans des modes de réalisation, l'instruction arithmétique comporte au moins une troisième source implicite dans le banc WRF partagé, 26W. Celle-ci peut être définie par le fil superviseur en écrivant dans le WRF. Ainsi le superviseur définit des poids communs à utiliser implicitement par les opérations arithmétiques de tous les fils de travail en cours d'exécution.

[0118] Les valeurs des adresses mémoire en paquets de trois ou "Tripacked" spécifiant $mAddr0:Addr0+1 dans le MRF, 26M, sont aussi mises à certaines valeurs initiales par le fil de travail respectif. Les valeurs de pas dans les champs de chaque registre de pas possible $mStride dans le MRF sont aussi définies par le fil de travail respectif. À chaque fois que l'instruction de chargement-stockage est exécutée, chacune des trois adresses (deux adresses de chargement et une adresse de stockage) dans les registres d'adresses mémoire $mAddr0:Addr0+1 est post-incrémentée de la valeur de pas respective. Voir de nouveau les figures 7 et 8. Par exemple, disons que l'instruction de chargement-stockage spécifie $mStride = $mStrideA et Strimm = 011000 (premier opérande de pas immédiat = 01, deuxième = 10 et troisième = 00). Cela signifie qu'après avoir réalisé le premier chargement, elle incrémente la première adresse de chargement de la valeur de pas dans le champ 0 de $mStrideA ; après avoir réalisé le

deuxième chargement elle incrémente la deuxième adresse de chargement de la valeur de pas dans le champ 1 de $mStrideA ; et après le stockage elle incrémente l'adresse de stockage de l'incrément atomique (incrément d'un pas d'adresse mémoire de 1 dans l'espace d'adresses qui est utilisé). Des instances ultérieures de l'instruction de chargement-stockage peuvent être définies pour spécifier la même valeur ou une valeur différente de $mStride, et les mêmes valeurs ou des valeurs différentes de chacun des trois opérandes immédiats. Ainsi, à chaque instruction de chargement-stockage de la série, les emplacements des chargements et du stockage peuvent être déplacés de manière contrôlable. Ainsi, le code peut efficacement balayer l'espace de données d'entrée de manière flexible sans avoir besoin d'instructions arithmétiques en entiers additionnelles pour calculer de nouvelles adresses.

[0119] La figure 9 illustre un exemple d'application dans lequel cela peut être utile, à savoir la réalisation d'une convolution d'un noyau K avec une portion multidimensionnelle de données d'entrée Xin. Par exemple les données d'entrée peuvent être un volume 3D de données d'entrée et le noyau K peut être un noyau 3D. Les données d'entrée peuvent aussi être appelées canaux d'entrée, et les données de sortie peuvent être appelées canaux de fonctionnalités de sortie. Lorsque l'on dit que les données sont multidimensionnelles, cela signifie que les données prennent une valeur pour chaque combinaison d'une pluralité de combinaisons de coordonnées dans un système de coordonnées à deux dimensions ou plus, chaque dimension (c'est-à-dire chaque axe) représentant une variable indépendante différente (et la valeur des données au niveau d'une combinaison donnée de coordonnées étant la variable dépendante).

[0120] Un exemple de cela pourrait être trouvé dans un traitement d'image où chaque valeur de données représente un pixel. Par exemple, typiquement 8 bits sont utilisés pour chacune de valeurs de Rouge, Vert et Bleu ; et ces valeurs entières de 8 bits vont ensuite être converties en valeur f16 avant l'opération de convolution. Chaque valeur R, G, B est traitée comme un plan d'entrée séparé (c'est-à-dire séparément pour chaque canal d'entrée). Ainsi chaque image d'entrée 2D a des dimensions x, y, z, les plans R, G, B remplissant l'axe z. Le volume de données peut par exemple comprendre une séquence de trames, deux dimensions du volume représentant les axes x et y spatiaux de la région de la trame et la troisième dimension représentant l'axe des temps correspondant à différentes trames dans la séquence. Dans un autre exemple, les trois dimensions du volume peuvent représenter trois dimensions x, y et z spatiales d'un modèle ou d'une image 3D d'un espace ou d'un objet. Dans un autre exemple, les données peuvent être seulement bidimensionnelles, représentant les axes x et y spatiaux d'une image fixe, l'axe des z représentant un nombre d'images d'entrée à traiter simultanément. On notera que l'applicabilité des techniques décrites n'est pas limitée à un traitement d'image ou un traitement graphique, et que des données ayant

une dimensionnalité supérieure ne sont aussi pas exclues. Par exemple, les deux, trois dimensions ou plus pourraient représenter différents aspects d'utilisateurs ou leurs circonstances dans une application d'apprentissage de comportement d'utilisateur, etc.

[0121] Le noyau K est une matrice de poids prédéterminée. Il peut représenter une certaine fonctionnalité pour la reconnaissance de laquelle le réseau neuronal est en cours d'apprentissage, ou qui est en train d'être recherchée pour une inférence ultérieure. Par exemple, dans le cas de données d'images le noyau peut représenter un bord ou une autre sorte de forme particulière. Dans une application d'apprentissage automatique, chaque nœud dans le réseau neuronal peut correspondre à un noyau respectif (bien que comme on va le décrire davantage ultérieurement, dans de nombreux réseaux neuronaux il y a de multiples nœuds 102 qui utilisent les mêmes valeurs de noyau mais qui ont juste des connexions 104 différentes). Dans des modes de réalisation, chaque nœud 102 est représenté par un fil de travail respectif, de sorte que chaque fil réalise une convolution respective d'un noyau K avec des données d'entrée respective de ce fil. Au moins certains des fils de travail peuvent utiliser le même noyau K dont les valeurs sont mémorisées dans le banc de registres de poids partagés (WRF) 26W, mais chaque fil de la sorte fait une convolution de ce noyau avec les propres données d'entrée respectives du fil, qui sont acheminées à partir de la mémoire 22 et vers celle-ci via son propre banc de registres (ARF), 26A, auxiliaire (ou arithmétique) respectif.

[0122] Une convolution discrète (*) dans trois dimensions peut être exprimée ainsi :

[0123] $Xin * K [x,y,z] = \Sigma_{x',y',z'} (K[x',y',z']Xin[x-x', y-y', z-z'])$

[0124] Une convolution comprend ainsi un balayage du noyau K systématiquement sur l'étendue de certaines ou de la totalité des positions possibles auxquelles le noyau peut être superposé avec les données d'entrée. À chacune de ces positions, chaque pixel (ou point de données) dans la partie des données d'entrée qui est superposée au noyau est multiplié par la valeur du noyau au point de superposition correspondant, et les résultats de toutes ces multiplications individuelles sont additionnés pour donner une sortie scalaire représentant la convolution du noyau avec les données d'entrée à cette position de superposition particulière. Cela est ensuite répété à différentes positions balayées du noyau par rapport aux données d'entrée Xin, afin de donner un volume correspondant de données de sortie (une valeur scalaire par position du noyau). Par exemple, dans l'exemple illustré en figure 9, le noyau K pourrait commencer placé dans un coin dans le volume des données d'entrée Xin, et chaque valeur du noyau est multipliée par le point de données auquel elle est superposée lorsque le noyau est dans cette position. Le noyau est ensuite déplacé d'un pas suivant la longueur du volume et les multiplications de points et la somme sont réalisées de nouveau, cette fois avec le noyau à la nouvelle position, etc. Lorsqu'on a fait balayer par le noyau la longueur complète du volume, il est décalé vers une nouvelle position de coordonnées et le

balayage dans la longueur est réalisé de nouveau, etc., jusqu'à ce que le volume entier ait été balayé. Le noyau à chaque position donnée génère un pixel (une valeur scalaire) des données de sortie. Par conséquent, les données de sortie Xout auront la même taille ou une taille similaire par rapport aux données d'entrée, en fonction de la taille de pas du balayage de convolution.

[0125] En pratique, cela doit être décomposé en opérations plus petites. Cela peut être réalisé en divisant la convolution des données avec le noyau à une position donnée en un certain nombre de produits scalaires vectoriels individuels et de sommes partielles. Cela est illustré dans les côtés du centre et de droite de la figure 9. Disons par exemple que le noyau a une taille de 3x3x16 pixels ("pixel" ne faisant pas nécessairement référence à des images pour l'objet de la présente description). C'est-à-dire qu'il est constitué de 3 par 3 par 16 valeurs de poids prédéterminées. Cela peut être décomposé en neuf vecteurs CW de 16 éléments (appelés aussi ici un noyau constituant du noyau K). Premièrement, le produit scalaire (c'est dire le produit scalaire) est effectué entre l'un de ces neuf vecteurs et les données d'entrée à la position correspondante, pour donner une somme partielle P (un scalaire). Ensuite le produit scalaire est pris entre un autre des neuf vecteurs et est accumulé avec la première somme partielle pour donner une deuxième somme partielle (aussi un scalaire). Ensuite le produit scalaire est effectué entre un autre des neuf vecteurs à 16 éléments et le résultat de cela est accumulé avec la deuxième somme partielle pour donner une troisième somme partielle, etc. Une fois que cela a été réalisé pour accumuler sur la totalité des neuf vecteurs, le résultat total de l'accumulation donne la valeur d'un seul point ou pixel dans les données de sortie Xout. Cela est ensuite répété à chaque position du noyau alors qu'on fait balayer le noyau K sur différentes positions relatives par rapport au volume de données d'entrée Xin, donnant ainsi un volume correspondant de données de sortie Xout (un point de données de sortie par position du noyau).

[0126] Dans des modes de réalisation, pour aider à réaliser des calculs comme des convolutions de manière efficace, le jeu d'instructions du processeur 4 comprend une instruction arithmétique ayant la forme d'une instruction de produit à accumulation ("amp"). Elle a la syntaxe suivante :

[0127] amp $aDst, $aSrcA, $aSrcB, Phase

[0128] où $aDst désigne de nouveau un opérande spécifiant une destination dans le banc ARF, 26A ; et $aSrcA, $aSrcB désignent des opérandes spécifiant deux sources dans le banc ARF. "Phase" est un opérande immédiat qui spécifie une phase de l'accumulation. Par exemple, dans des modes de réalisation l'opérande de phase est constitué de deux bits spécifiant l'une de quatre phases possibles 0…3. Dans des modes de réalisation, l'instruction amp peut être dénommée f16v4sisoamp, en référence au fait qu'elle prend un vecteur de quatre valeurs en virgule flottante à demi-

précision (16 bits) comme première entrée, et une valeur en virgule flottante à simple précision (32 bits) comme deuxième entrée ; et fournit une valeur en virgule flottante à simple précision (32 bits) (“siso” faisant référence à "entrée en simple précision, sortie en simple précision"). En fait dans des modes de réalisation la deuxième entrée est une paire de deux valeurs en virgule flottante à simple précision (2 x 32 bits), comme l'est aussi la sortie. Toutefois on notera que ces valeurs particulières de précision et le nombre d'éléments ne sont pas limitatifs pour tous les modes de réalisation possibles.

[0129] Pour un ou plusieurs types d'instructions comme l'instruction amp, dans des modes de réalisation, l'emplacement des poids dans le banc WRF est complètement implicite. En variante ou en plus, un ou plusieurs autres types d'instructions peuvent prendre un opérande additionnel (non représenté) spécifiant quel ensemble de poids il faut utiliser parmi quelques ensembles différents dans le WRF. Un exemple peut être l'instruction de convolution mince (“slic”) mentionnée dans la suite.

[0130] Combinée avec l'instruction de chargement-stockage décrite précédemment (ld2xst), une instruction comme l'instruction amp peut être utilisée pour acheminer de manière efficace des données à partir de la mémoire, réaliser les produits scalaires et les sommes partielles, et acheminer et remettre les sommes partielles en mémoire. Considérons par exemple un programme comprenant une séquence en boucle de quatre paires d'instructions, de la manière suivante.

[0131] Loop {

[0132] ldx2st Xin-Pin, Pout, Tripacked, Strides; amp Pout, Xin, Pin, Phase=0;

[0133] ldx2st Xin-Pin, Pout, Tripacked, Strides; amp Pout, Xin, Pin, Phase=1;

[0134] ldx2st Xin-Pin, Pout, Tripacked, Strides; amp Pout, Xin, Pin, Phase=2;

[0135] ldx2st Xin-Pin, Pout, Tripacked, Strides; amp Pout, Xin, Pin, Phase=3;

[0136] }

[0137] Un autre exemple de version serait :

[0138] Loop {

[0139] ldx2st Xin-Pin, Pout_A, Tripacked, Strides; amp Pout_A, Xin, Pin, Phase=0;

[0140] ldx2st Xin-Pin, Pout_B, Tripacked, Strides; amp Pout_B, Xin, Pin, Phase=1;

[0141] ldx2st Xin-Pin, Pout_A, Tripacked, Strides; amp Pout_A, Xin, Pin, Phase=2;

[0142] ldx2st Xin-Pin, Pout_B, Tripacked, Strides; amp Pout_B Xin, Pin, Phase=3;

[0143] }

[0144] etc.

[0145] Ici encore, dans des modes de réalisation chaque paire est un groupe d'instructions, c'est-à-dire des instructions exécutées simultanément en suivant des pipelines respectifs, par exemple les pipelines principal et auxiliaire.

[0146] Chaque paire d'instructions comprend une instance de l'instruction de chargement-stockage suivie d'une instance de l'instruction de produit avec accumulation (amp).

Les sources de l'instruction amp sont la destination des deux chargements réalisés par l'instruction de chargement-stockage dans la même paire d'instructions ou une paire précédente. Dans des modes de réalisation qui exécutent chaque paire sous forme d'un groupe de deux instructions simultanées, les sources de l'instruction amp (Xin, Pin) sont définies comme étant les mêmes que les destinations du chargement réalisé dans le groupe précédent. La destination de l'instruction amp est la source du stockage réalisé par l'instruction de chargement-stockage dans une paire d'instructions ultérieure, par exemple le groupe suivant. L'une des sources $aSrcA, $aSrcB de l'instruction amp est utilisée pour prendre un vecteur d'entrée x à partir des données d'entrée Xin. L'autre est utilisé pour prendre une somme partielle. L'instruction amp, lorsqu'elle est exécutée, réalise un produit scalaire de son vecteur d'entrée x et d'un vecteur de poids CW correspondant provenant du banc de registres de poids WRF, 26W (ce vecteur CW étant un noyau constituant du noyau global ou 3D K). L'opérande de phase spécifie une phase dans le but d'accumuler les résultats des produits scalaires. Dans chaque paire d'instructions de la séquence dans une instance donnée de la boucle, l'opérande de phase est mis à une valeur respective différente spécifiant une phase successive différente de la séquence. Dans ces phases respectives dans la séquence spécifiée par l'opérande de phase, l'effet de l'instruction amp est d'accumuler les résultats de tous les produits scalaires successifs. L'accumulation se fait dans la première phase de la séquence avec une somme partielle d'entrée.

[0147] La fonctionnalité de pas de progression de l'instruction de chargement-stockage permet au programme de se décaler automatiquement suivant les adresses mémoire vers l'élément de données suivant à chaque paire d'instructions, sans avoir besoin d'instructions arithmétiques en entiers séparées pour calculer de nouvelles adresses mémoire à inclure dans le banc MRF, 26M. La capacité de l'instruction de chargement-stockage décrite à sélectionner l'un quelconque d'un certain nombre de pas préprogrammés dans le registre de pas $mStride est particulièrement utile pour gérer des données multidimensionnelles comme cela est illustré à titre d'exemple en figure 9. L'espace mémoire est monodimensionnel mais les données peuvent avoir deux, trois dimensions ou même plus. En fonction du mappage de l'espace mémoire 1D vers l'espace de données multidimensionnel, ensuite lorsque le noyau K est balayé à travers les données d'entrée Xin, il peut être parfois nécessaire de faire des pas de tailles différentes à travers les adresses mémoire correspondantes dans l'espace mémoire. Par exemple lorsque le noyau K est balayé sur la longueur suivant une dimension des données d'entrée Xin, alors l'adresse mémoire des données à chaque position balayée peut s'incrémenter par pas atomiques, mais lorsque le balayage doit se décaler dans le plan perpendiculaire, alors un pas de taille différente dans les adresses mémoire peut être nécessaire pour obtenir l'élément ou le sous-ensemble suivant des

données d'entrée Xin (et indépendamment, le même processus est réalisé pour les deuxièmes adresses d'entrée et de sortie. Les différents champs dans le registre de pas $mStride, ou les registres de pas (par exemple $mStrideA, $mStrideB, ...) peuvent avantageusement être préprogrammés avec des sauts de tailles différentes dans l'espace mémoire. Le programmeur ou le compilateur peut alors définir les opérandes immédiats Strimm pour sélectionner la taille de saut qu'il faut utiliser pour chaque instruction de chargement-stockage (sans avoir besoin d'instructions arithmétiques en entiers séparées pour recalculer l'adresse suivante ou la taille de pas suivante).

[0148] Dans des modes de réalisation, l'instruction amp est une instruction de produit matriciel avec accumulation. On va décrire le fonctionnement de cela en faisant référence aux figures 10 à 12. Comme cela est représenté en figure 10, l'instruction amp, lorsqu'elle est exécutée par le FPU, réalise une multiplication d'une matrice MxN de poids (provenant du WRF, 26W) par un vecteur d'entrée à N éléments provenant des données d'entrée Xin. Dans des modes de réalisation M=8 et N=16. On notera que chacune des M rangées de la matrice correspond à un vecteur constituant provenant d'un noyau respectif différent K0, K1....KM-1. Par conséquent l'instruction amp est utilisée pour réaliser une composante de la convolution de M noyaux K différents avec les données d'entrée en parallèle. Par exemple chacun de ces noyaux peut correspondre à une convolution d'un élément caractéristique respectif différent (par exemple un bord ou une forme différente) avec les données d'entrée. Dans le cas d'un réseau neuronal de convolution il y a des filtres d'éléments caractéristiques, chacun des M noyaux tentant de détecter la présence d'un élément caractéristique différent dans les couches d'entrée. Pour un noyau K donné, l'instruction amp réalise seulement un produit scalaire et une accumulation. Toutefois, la matrice permet de réaliser de manière efficace des convolutions de multiples noyaux avec les données en parallèle.

[0149] Le vecteur d'entrée à N éléments est séparé en N1 segments, chacun comprenant un sous-vecteur de N2 éléments (ainsi N=N1xN2). Dans des modes de réalisation N1=4 et N2=4. L'opérande de phase de l'instruction amp prend l'une de N1 valeurs possibles différentes 0....N1-1 correspondant à N1 segments différents. Dans chaque instance de la séquence bouclée, l'instruction amp dans chaque paire successive prend une valeur différente de l'opérande de phase spécifiant une phase successive différente parmi les phases 0....N1-1. Chaque phase réalise le produit scalaire de l'un correspondant des N1 sous-vecteurs du vecteur d'entrée à N éléments, et de chacune des M rangées de la matrice (et ainsi une partie du calcul pour chacun des M noyaux K différents). Le FPU dans l'unité d'exécution 18 (dans des modes de réalisation se trouvant dans l'unité d'exécution auxiliaire ou arithmétique 18A) comporte M états d'accumulateur $AACC (par fil d'exécution). Dans des modes de réalisation ceux-ci sont mis en œuvre sous forme d'états internes 57 du FPU. Toutefois, dans des variantes de réalisation il n'est

pas exclu qu'ils puissent être mis en œuvre sous forme de registres dans l'un des bancs de registres 26 (par exemple le banc ARF 26A du fil respectif). Dans l'exemple illustré ce sont les états à numéros pairs $AACC[0], $AACC[2]… $AACC[14] ; tandis que les états à numéros impairs $AACC[1], $AACC[3]… $AACC[15] peuvent être utilisés pour propager des valeurs entre des boucles différentes de la séquence, mais cela est un détail de mise en œuvre et n'est pas limitatif.

[0150] La figure 11 illustre schématiquement le fonctionnement de l'instruction amp. La figure 12 donne un exemple d'une série d'instructions amp. Il faut comprendre que chacune est précédée d'une instruction de chargement-stockage correspondante d'une paire d'instructions, comme cela a été présenté précédemment. En figure 12, Pn est une somme partielle n d'entrée à simple précision, xn est un vecteur d'entrée f16v4, CWm,n est l'état de poids commun $CWEI_m_n, et Rn est le résultat final à simple précision d'accumulations de produits scalaires successifs qui ont commencé avec Pn.

[0151] En fonctionnement, en réponse au code opération (décodé) de l'instruction amp, le FPU de l'unité d'exécution 18 (dans des modes de réalisation l'unité d'exécution auxiliaire 18A) réalise les opérations suivantes.

[0152] - À partir du ou des registres se trouvant dans le banc ARF 26A spécifiés par l'un des opérandes de source de l'instruction amp, elle prend un nombre M/N1 de sommes partielles et les place temporairement dans un état de propagation pour la boucle suivante. Cet état de propagation pourrait être mis en œuvre dans l'état interne 57 du FPU, ou dans des variantes de réalisation ce pourrait être un autre registre dans l'un des bancs de registres 26 (par exemple le banc ARF). Chaque phase prend un sous-ensemble différent de M/N1 sommes partielles. Sur les N1 phases, M sommes partielles de la sorte sont prises dans l'état de propagation. Ainsi si M=8 et N1=4, alors chaque instruction amp prend deux sommes partielles (par exemple deux valeurs f32) comme entrées et maintient celles-ci dans l'état de propagation, et sur toutes les quatre phases ensuite huit somme partielles sont reçues.

[0153] - À partir du ou des registres se trouvant dans le banc ARF 26A spécifié par l'un des opérandes de source de l'instruction amp, elle prend un segment de N2 éléments (sous-vecteur) du vecteur d'entrée xin. Pour chacune des M rangées de la matrice (et ainsi chacun des M noyaux K0…KM-1), elle réalise un produit scalaire de cela avec un sous-vecteur correspondant de poids CW provenant du banc WRF. Le choix de la colonne dans laquelle le sous-vecteur de poids est pris dépend de la valeur de l'opérande de phase. En outre, si la phase est la première phase (c'est-à-dire la phase initiale) dans la séquence, alors une somme correspondante parmi les sommes partielles est aussi ajoutée à chacun des M produits scalaires. On notera que puisque chaque instruction amp dans la boucle précédente a pris M/N1 sommes partielles respectives, où N1 est le nombre de phases, alors la boucle précédente aura placé toutes

les M sommes partielles dans l'état de propagation prêtes pour la boucle courante.

[0154] - Pour chacune des M rangées, le résultat du calcul susmentionné est accumulé (additionné) avec toute valeur se trouvant déjà dans un état correspondant des M états d'accumulateur $AACC, générant ainsi l'un correspondant de M résultats R0...RM-1. Sur les N1 phases, l'instruction amp va ainsi réaliser un produit scalaire du vecteur d'entrée à N éléments complet avec un vecteur de poids à N éléments correspondant provenant du banc WRF.

[0155] - Dans la boucle courante, un nombre M/N1 de résultats R de la boucle précédente est fourni au ou aux registres de destination dans le banc ARF spécifié par l'opérande de destination de l'instruction amp. Chaque phase produit un sous-ensemble différent de M/N1 sommes partielles, et sur les N1 phases, M sommes partielles de la sorte sont prises dans l'état de propagation. Ainsi dans des modes de réalisation où M=8 et N1=4, deux résultats sont fournis par instruction amp (par exemple deux f32s), et tous les huit résultats R0...R7 sont fournis sur toutes les quatre phases.

[0156] - Après la dernière phase de la boucle courante (ou au démarrage de la première phase de la boucle suivante avant que des résultats soient écrasés), les résultats R de la boucle courante sont placés temporairement dans un état de propagation prêts à être fournis dans la boucle suivante. Cet état de propagation pourrait être mis en œuvre dans l'état interne 57 du FPU, ou dans des variantes de mise en œuvre ce pourrait être un autre registre dans l'un des bancs de registres 26 (par exemple le banc ARF).

[0157] Pour résumer, une seule somme partielle est : (1) ajoutée au premier résultat de produit scalaire et mémorisée dans l'état d'accumulateur (écrasant la valeur contenue précédemment), (2) accumulée avec trois résultats supplémentaires de produits scalaires, et (3) copiée dans l'état de propagation temporaire (auquel cas cette valeur va être écrite dans l'opérande de destination d'une instruction amp future). Une alternative à l'étape (3) est à la place d'écrire la somme partielle résultante directement sortie de l'accumulateur dans l'opérande de destination de l'instruction amp. Dans tous les cas, les étapes (1) et (3) peuvent être superposées dans le matériel, assurant une boucle de 4 cycles.

[0158] Dans des modes de réalisation l'état d'accumulateur n'a pas besoin d'être réinitialisé explicitement. Puisque le point de démarrage pour chaque multiplication matrice-vecteur est la somme partielle d'entrée lue dans la mémoire, le volume de sortie a tous les éléments mis à zéro au début de la convolution, plutôt que de réinitialiser l'état d'accumulateur. C'est-à-dire que, dans des modes de réalisation, le pavé 4 permet en effet qu'on initialise à zéro la totalité de l'état d'accumulateur.

[0159] On notera que l'état de propagation d'entrée de l'instruction amp courante est maintenu pour être utilisé dans la boucle suivante. Aussi, le résultat fourni par l'instruction amp au registre de destination dans le banc ARF est l'état d'accumulateur

existant avant que l'opération d'accumulation de l'instruction amp courante soit appliquée - c'est le résultat d'une précédente séquence courte d'accumulations, qui n'inclut pas l'entrée courante.

[0160] Le programme est agencé (par le programmeur ou le compilateur) de manière à ce que les résultats R deviennent les entrées de sommes partielles P pour des boucles ultérieures. La relation exacte est laissée du ressort du programmeur ou du compilateur et dépend de la structure des données et du noyau. Typiquement, les sommes partielles auront une valeur initiale (en mémoire) de 0.

[0161] On notera qu'en raison du décalage temporel entre les boucles, la séquence nécessite une période de mise en action. Il y a une latence entre un vecteur d'entrée particulier qui est présenté et le résultat du calcul à 4 phases qui est renvoyé par l'instruction amp. Cette latence est fixe et signifie que les M premiers résultats ne sont pas valides, sont sans intérêt ou sans signification, et sont supprimés dans le code de mise en action. Après cette période de mise en action dans laquelle aucun résultat utile n'est produit, le code rentre alors dans la boucle répétitive qui va de l'avant et écrit les résultats utiles dans les emplacements mémoire requis (64 bits à la fois dans l'exemple illustré). Il y a aussi une période de récupération après la boucle interne dans laquelle aucune nouvelle entrée n'est fournie mais dans laquelle les valeurs de sortie finales sont mémorisées. La figure 12 illustre la période de mise en action et la "latence de résultats" pour f16v4sisoamp.

[0162] Dans des modes de réalisation, le nombre de noyaux est M = 8, le nombre d'éléments dans le vecteur d'entrée est N = 16, et le nombre de phases est N1 = 4. Dans des modes de résolution, chaque élément du vecteur d'entrée est une valeur f16, et chaque somme partielle d'entrée et chaque résultat est une valeur f32. Dans de tels modes de réalisation, le processus susmentionné peut charger et traiter des données à une cadence de deux sommes partielles f32 et quatre éléments f16 du vecteur d'entrée par paire d'instructions, et peut fournir et mémoriser des données à une cadence de deux valeurs f32 par paire d'instructions. C'est-à-dire 128 bits d'entrée et 64 bits de sortie par paire d'instructions. Cela correspond aux deux ports de chargement d'une largeur de 64 bits et au port de stockage d'une largeur de 64 bits entre le LSU 55 et la mémoire de données 22 (voir de nouveau la figure 4). Aussi, en raison de la mise en paquets de trois des adresses, le processus peut être pris en charge via deux ports d'une largeur de 32 bits à partir du banc MRF 26M vers le LSU, plus un autre pour les pas. Deux ports d'une largeur de 64 bits sont utilisés à partir du banc ARF 26A vers le FPU dans l'unité d'exécution auxiliaire 18A. La figure 12 montre que pour chaque vecteur d'entrée de 16 éléments, la boucle produit 8 valeurs de sortie (1 valeur de sortie par noyau). Dans ce scénario particulier, ces valeurs de sortie comportent 32 bits chacune. La boucle susmentionnée traite 16 valeurs d'entrée tous les 4 coups (4 valeurs d'entrée

f16 à chaque instruction). Pour soutenir cette cadence d'entrée, le processeur 4 par conséquent doit produire 8 valeurs de 32 bits à chaque boucle de 4 coups - ce qui donne un montant de 4 stockages de 64 bits par boucle, ou un stockage de 64 bits par coup.

[0163] Les poids CW sont pris dans le banc WRF partagé 26W et sont les mêmes pour tous les fils d'exécution. Il se trouve que dans des applications comme des réseaux neuronaux, il y a beaucoup de scénarios où il est nécessaire de multiplier (comme en tant que partie d'une convolution) le même ensemble de poids par des données différentes. Par exemple, certains nœuds dans un réseau neuronal peuvent comprendre exactement les mêmes poids mais avoir des connexions différentes. Par exemple chaque noyau K(m=0...M) peut représenter une convolution d'un élément caractéristique (par exemple un bord ou une forme) avec les données d'entrée (par exemple une région ou un volume de pixels graphiques). Par conséquent, les fils de travail correspondant à ces noyaux n'ont pas besoin de poids séparés. Au lieu de cela, il est prévu seulement une copie de l'état de poids que possède le superviseur. L'utilisation d'opérandes de fils partagés provenant d'un banc de registres de poids partagés a ainsi avantageusement besoin de moins d'espace de registre pour les poids. La figure 12 montre que pour chaque vecteur d'entrée de 16 éléments, la boucle produit 8 valeurs de sortie (1 valeur de sortie par noyau). Dans ce scénario particulier, ces valeurs de sortie ont 32 bits chacune. La boucle susmentionnée travaille sur 16 valeurs d'entrée tous les 4 cycles (4 valeurs d'entrée f16 à chaque instruction). Pour soutenir cette cadence d'entrée, le processeur 4 produit par conséquent 8 valeurs de 32 bits à chaque boucle de 4 cycles - ce qui donne un montant de 4 stockages de 64 bits par boucle, ou un stockage de 64 bits par cycle.

[0164] Dans des modes de réalisation, les accumulateurs impairs $AACC[1], $AACC[3], ..., $AACC[15] sont utilisés pour échelonner les résultats (Rx), et pour faire la même chose aux entrées de sommes partielles (Px). Les entrées partielles sont fournies par deux à la fois aux moteurs à l'arrière - dans des registres accumulateurs $AACC[13] et $AACC[15]. À chaque cycle ces valeurs se déplacent vers l'avant, et donc dans ce cas dans $AACC[11] et $AACC[9]. Lorsqu'elles atteignent l'unité AMP de destination, elles se déplacent dans les accumulateurs à numéros pairs, plutôt que dans ceux à numéros impairs, prêtes pour l'accumulation. Ainsi {$AACC[15], $AACC[13]} -> {$AACC[11], $AACC[9]} -> {$AACC[7], $AACC[5]} -> {$AACC[2], $AACC[0]}.

[0165] Les figures 13 et 14 illustrent un autre type d'instruction arithmétique qui peut profiter d'une instruction de chargement-stockage du type décrit ici, à savoir une instruction de convolution qui effectue une convolution des données d'entrée acheminées via le banc ARF 26A du fil respectif avec un ensemble 2D de poids provenant du banc de registres de poids commun (WRF) 26W. Dans des modes de réalisation, cette ins-

truction peut prendre la forme d'une convolution mince d'un vecteur en virgule flottante à demi-précision (instruction slic), par exemple "f16v4slic" faisant référence au fait qu'elle opère sur des sous-vecteurs de 4 éléments de valeurs f16. Elle a une entrée à demi-précision et une entrée à simple précision, et une sortie à simple précision. slic est similaire et utilise le même matériel que l'instruction amp. Toutefois cette instructions réalise une convolution 1 x 4 x 4 (qui est une vraie convolution, pour chacun de 2 noyaux, plutôt qu'un produit scalaire à accumulation 1 x 1 x 16 pour chacun de 8 noyaux. En fait, dans des modes de réalisation elle peut être agencée pour réaliser une convolution 1 x N x 4, où N vaut 2, 3 ou 4.

[0166] Les instructions de produit matriciel à accumulation ("amp") et de convolution mince ("slic") facilitent des séquences de multiplications et accumulations à hautes performances, pour des scénarios où un partage de poids entre des contextes de fils de travail est approprié. Dans des modes de réalisation les instructions amp et slic peuvent être supportées à la fois pour le format de nombres à simple précision et le format à demi-précision et fonctionnent de la même manière de base. L'état de configuration de calcul commun (comprenant les poids partagés dans le WRF 26W) est d'abord initialisé par le contexte superviseur. Deux flux de données d'entrée sont ensuite traités : les données d'activation d'entrée (pixels), et les valeurs de sommes partielles spécifiant une valeur de départ pour une séquence de multiplications et d'accumulations ultérieure (pour des convolutions ce sont les pixels/activations de sortie calculés partiellement). Un seul flux de données de sortie est produit : les valeurs de sommes partielles accumulées résultantes. Chaque valeur d'entrée de somme partielle est soumise à une séquence de longueur fixe d'opérations de multiplication-accumulation, avant que le résultat de somme partielle final soit présenté comme sortie. De nombreuses opérations de produits scalaires et d'accumulations se déroulent en parallèle, réalisées par des moteurs de calcul. Le premier multiplicande est fourni par le flux de données d'entrée et le deuxième est fourni par l'état de configuration de calcul commun.

[0167] Le tableau suivant liste certains exemples de variantes des instructions amp et slic, dont certaines ou la totalité peuvent être incluses dans le jeu d'instructions du processeur dans des modes de réalisation.

[0168]

[Tableaux1]

Instruction	Format de données d'entrée	Format de somme partielle d'entrée	Format de somme partielle de sortie
f16v4sisoamp	Demi-précision (v4)	Simple précision (v2)	Simple précision (v2)
f16v4hihoamp	Demi-précision (v4)	Demi-précision (v2)	Demi-précision (v2)
f16v4sihoamp	Demi-précision (v4)	Simple précision (v2)	Demi-précision (v2)
f32sisoamp	Simple précision (scalaire)	Simple précision (v2)	Simple précision (v2)
f16v4sisoslic	Demi-précision (v4)	Simple précision (v2)	Simple précision (v2)
f16v4hihoslic	Demi-précision (v4)	Demi-précision (v2)	Demi-précision (v2)
f16v4sihoslic	Demi-précision (v4)	Simple précision (v2)	Demi-précision (v2)
f32sisoslic	Simple précision (scalaire)	Simple précision (v2)	Simple précision (v2)

[0169] On notera que pour certains types d'instructions l'emplacement des poids est complètement implicite, mais que pour d'autres types elles prennent un opérande sélectionnant celui parmi un certain nombre d'ensembles différents qu'il faut utiliser dans le WRF. L'instruction f16v4sisoslic par exemple ne nécessite qu'un quart de l'état de poids de f16v4sisoamp. Il en résulte que le logiciel est autorisé à pré-charger jusqu'à 4 ensembles de poids. La sélection d'ensemble de poids est spécifiée par deux bits d'un opérande immédiat. Par contre, pour f16v4sisoamp, on prend l'état de poids complet et ainsi il n'y a pas de sélection à l'intérieur de celui-ci.

[0170] On notera que les modes de réalisation susmentionnés ont été décrits seulement à titre d'exemple.

[0171] Par exemple, le domaine de la présente description est l'architecture décrite précédemment dans laquelle un contexte séparé est prévu pour le fil superviseur, ou dans laquelle le fil superviseur s'exécute dans un créneau puis abandonne son créneau à un fil de travail. Le superviseur pourrait à la place utiliser un contexte d'usage général. Ou dans un autre agencement par exemple, le superviseur pourrait s'exécuter dans son propre créneau dédié. En outre, la mise en œuvre n'est pas limitée au fait qu'un fil

spécifique parmi les fils d'exécution ait un rôle de superviseur. En outre, le domaine de la description n'est pas limité au fait que le processeur 4 soit un pavé dans une matrice de pavés. Dans des variantes de réalisation le processeur 4 pourrait par exemple être un processeur autonome ou un processeur en une seule puce.

[0172] La mise en œuvre de l'unité d'exécution 18 n'est pas limitée au fait que l'on ait une unité d'exécution principale 18M et une unité d'exécution auxiliaire 18A séparées ; ni à des bancs MRF et ARF séparés. En général, les registres pourraient appartenir à l'un quelconque d'un ou plusieurs bancs de registres, qui pourraient être partagés entre des opérations d'accès mémoire et arithmétiques différentes, ou séparées.

[0173] Les instructions arithmétiques qui utilisent le banc de registres de poids partagés ne sont pas limitées à des instructions de produit matriciel ou de convolution. De manière plus générale le banc de registres de poids partagés peut être utilisé pour n'importe quelle type d'instruction arithmétique ou n'importe quelle combinaison d'instructions, par exemple une instruction de produit matriciel sans accumulation dans laquelle l'accumulation est faite dans des instructions séparées ou par du code d'usage général, ou d'autres séquences de multiplications matricielles autres que des convolutions, où les résultats de produits précédents ne sont pas nécessairement les entrées de produits ultérieurs. D'autres exemples peuvent inclure des instructions de produit scalaire vectoriel ou de produit scalaire vectoriel à accumulation - c'est-à-dire où les instructions arithmétiques n'ont pas à appliquer M noyaux différents en parallèle.

[0174] En outre, le domaine de la présente description n'est pas limité aux tailles de registres, aux largeurs en bits de ports, aux nombres de ports, aux précisions de valeurs, aux tailles de vecteurs ou aux tailles de matrices particulières décrites ci-avant à titre d'exemple. D'autres largeurs en bits de registres et de ports sont possibles, comme le sont d'autres précisions de valeurs, et d'autres tailles de vecteurs ou de matrices (par exemple en ce qui concerne le nombre d'éléments). En outre, dans d'autres mises en œuvre, d'autres manières d'empaqueter les deux adresses de chargement et l'adresse de stockage dans deux registres sont possibles, par exemple trois adresses de 10 bits dans deux registres de 16 bits, ou trois adresses de 42 bits dans deux registres de 64 bits. Aussi, les adresses de chargement et de stockage ne doivent pas nécessairement avoir la même longueur. En outre, d'autres manières d'empaqueter les valeurs de pas dans le registre de pas sont aussi possibles dans des variantes de réalisation, par exemple quatre champs de pas de 8 bits dans un registre de 32 bits, etc.

[0175] Dans encore d'autres variantes, l'applicabilité de la description n'est pas limitée au traitement d'image. Le terme "noyau" tel qu'il est utilisé ici pourrait désigner toute matrice de poids appliquée en tant que partie d'un processus quelconque comprenant des opérations comme des multiplications vectorielles, des multiplications matricielles ou des convolutions, ou autres (par exemple des corrélations). Aussi l'applicabilité

n'est pas limitée à un volume de données 3D. Les données d'entrée (et le noyau) pourraient être linéaires ou 2D, ou avoir une multi-dimensionnalité plus grande (>3 variables indépendantes ou degrés de liberté). En outre, le domaine de la description n'est pas limité à des applications d'apprentissage automatique. Il existe de nombreuses autres applications où l'on peut souhaiter réaliser une opération sur une combinaison d'un ou plusieurs opérandes qui sont communs à des fils d'exécution simultanés et d'un ou plusieurs opérandes qui sont spécifiques à chaque fil.

[0176] D'autres variantes ou d'autres cas d'utilisation des techniques décrites peuvent apparaître à l'homme de l'art une fois donnée la description faite ici. Le domaine de la description n'est pas limité par les modes de réalisation décrits mais seulement par les revendications jointes.

Revendications

[Revendication 1] Processeur comprenant :
- un ou plusieurs bancs de registres ; et
- une unité d'exécution agencée pour exécuter des instances de types d'instructions définis dans un jeu d'instructions, chaque instruction se trouvant dans le jeu d'instructions étant constituée d'un code opération et d'un ou plusieurs opérandes ;
dans lequel l'unité d'exécution est une unité d'exécution à fils d'exécution en barillet agencée pour exécuter une pluralité de fils d'exécution simultanés chacun dans un créneau temporel respectif différent d'une séquence répétitive de créneaux temporels entrelacés, et pour chacun des fils d'exécution simultanés, lesdits un ou plusieurs bancs de registres comprennent un ensemble respectif de registres de contexte agencé pour contenir un état de programme du fil d'exécution respectif, chaque ensemble de registres de contexte comprenant un ensemble respectif de registres d'opérandes arithmétiques destiné à être utilisé par le fil d'exécution respectif ;
dans lequel l'un desdits un ou plusieurs bancs de registres comprend en outre un ensemble de registres de poids partagés commun à certains ou à la totalité des fils d'exécution simultanés ;
dans lequel les types d'instructions définis dans le jeu d'instructions comprennent une instruction arithmétique ayant des opérandes spécifiant une source et une destination parmi l'ensemble respectif de registres arithmétiques du fil dans lequel l'instruction arithmétique est exécutée ; et
dans lequel l'unité d'exécution est agencée de manière à, en réponse au code opération de l'instruction arithmétique, réaliser une opération comprenant la multiplication d'une entrée provenant de ladite source par au moins l'un des poids provenant d'au moins l'un des registres de poids partagés, et placer un résultat dans ladite destination.

[Revendication 2] Processeur selon la revendication 1, dans lequel ledit au moins l'un des registres de poids partagés est implicite d'après le code opération de l'instruction arithmétique, n'étant spécifié par aucun opérande de l'instruction arithmétique.

[Revendication 3] Processeur selon la revendication 1, dans lequel l'instruction arithmétique prend un autre opérande spécifiant ledit au moins l'un des registres de poids partagés parmi l'ensemble de registres de poids

partagés.

[Revendication 4] Processeur selon l'une quelconque des revendications précédentes, dans lequel l'entrée comprend un vecteur, et la multiplication comprend un produit scalaire de l'entrée avec un vecteur de poids provenant des registres de poids partagés.

[Revendication 5] Processeur selon les revendications 3 et 4, dans lequel ledit au moins l'un des registres de poids partagés comprend un sous-ensemble des registres de poids partagés pris parmi une pluralité de sous-ensembles, chaque sous-ensemble contenant un vecteur de poids respectif ; et dans lequel l'autre opérande sélectionne à partir duquel sous-ensemble il faut prendre le vecteur de poids à utiliser dans la multiplication.

[Revendication 6] Processeur selon l'une quelconque des revendications précédentes, dans lequel l'instruction arithmétique comprend une instruction parmi : une instruction de produit scalaire vectoriel, une instruction de produit scalaire vectoriel à accumulation, une instruction de produit matriciel, une instruction de produit matriciel à accumulation, ou une instruction de convolution.

[Revendication 7] Processeur selon l'une quelconque des revendications précédentes, dans lequel les fils d'exécution simultanés comprennent une pluralité de fils de travail et l'unité d'exécution est en outre agencée pour exécuter, au moins à certains instants, un sous-programme superviseur comprenant au moins un fil superviseur agencé pour gérer les fils de travail.

[Revendication 8] Processeur selon la revendication 7, dans lequel le sous-programme superviseur est agencé pour écrire les poids dans le banc de registres de poids partagés.

[Revendication 9] Processeur selon la revendication 8, agencé de telle sorte que les poids dans les registres de poids partagés peuvent être écrits seulement par le sous-programme superviseur, et les fils de travail peuvent seulement lire les registres de poids partagés.

[Revendication 10] Processeur selon l'une quelconque des revendications 7 à 9, dans lequel les registres de contexte comprennent un ensemble respectif des ensembles de registres de contexte pour chacun des fils de travail qui peuvent être exécutés simultanément, et un ensemble additionnel de registres de contexte agencé pour contenir un état de programme du sous-programme superviseur.

[Revendication 11] Processeur selon la revendication 10, dans lequel le sous-programme superviseur est agencé pour commencer en s'exécutant initialement dans tous les créneaux, et pour écrire les poids avant de lancer les fils de

travail ; et dans lequel le sous-programme superviseur lance chacun des fils de travail en abandonnant chacun de certains ou de la totalité des créneaux dans lesquels le sous-programme superviseur s'exécute initialement à certains fils respectifs parmi les fils de travail.

[Revendication 12] Processeur selon la revendication 11, dans lequel le jeu d'instructions comprend une instruction d'exécution qui, lorsqu'elle est exécutée en tant que partie du sous-programme superviseur, amène le créneau dans lequel l'instruction d'exécution est exécutée à être abandonné à l'un des fils de travail de sorte que le fil de travail est lancé dans ce créneau à la place du sous-programme superviseur.

[Revendication 13] Processeur selon la revendication 12, dans lequel le jeu d'instructions comprend une instruction de sortie qui, lorsqu'elle est exécutée en tant que partie de l'un des fils de travail, amène le créneau dans lequel l'instruction de sortie est exécutée à être rendu au sous-programme superviseur de sorte que le sous-programme superviseur continue à s'exécuter dans ce créneau de nouveau à la place du fil de travail.

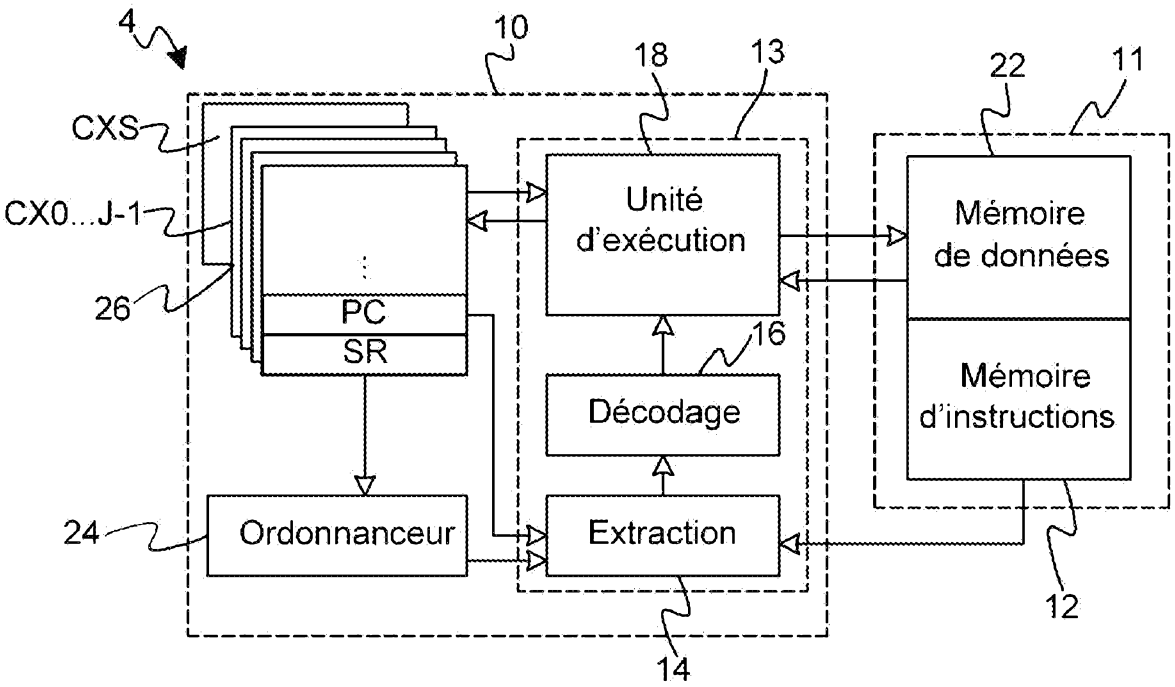
[Revendication 14] Processeur selon l'une quelconque des revendications 7 à 13, dans lequel les bancs de registres comprennent un banc de registres arithmétiques séparé pour chaque fil de travail simultané, le banc de registres arithmétiques respectif comprenant les registres d'opérandes arithmétiques respectifs.

[Revendication 15] Processeur selon la revendication 14, dans lequel les bancs de registres comprennent un banc de registres de poids séparé comprenant les registres de poids.

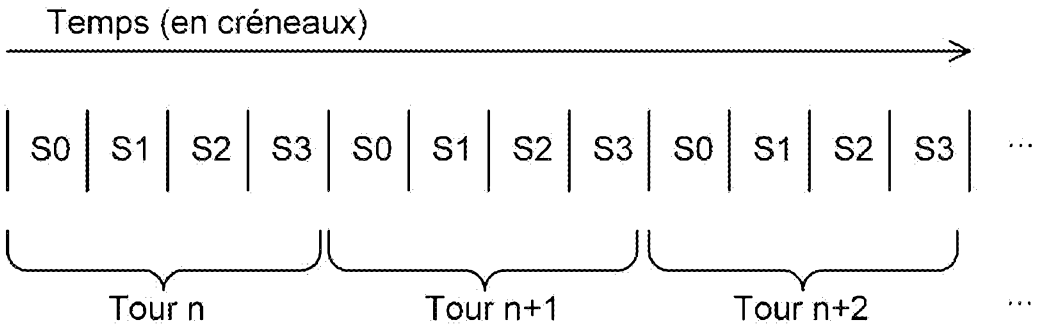
[Revendication 16] Processeur selon la revendication 15, dans lequel le banc de registres de poids est agencé de telle sorte qu'il peut être écrit seulement par le sous-programme superviseur et les fils de travail peuvent seulement lire le banc de registres de poids.

[Revendication 17] Procédé d'actionnement d'un processeur agencé selon l'une quelconque des revendications 1 à 16, le procédé comprenant l'exécution d'un programme comprenant une ou plusieurs instances de l'instruction arithmétique sur le processeur par l'intermédiaire de l'unité d'exécution.

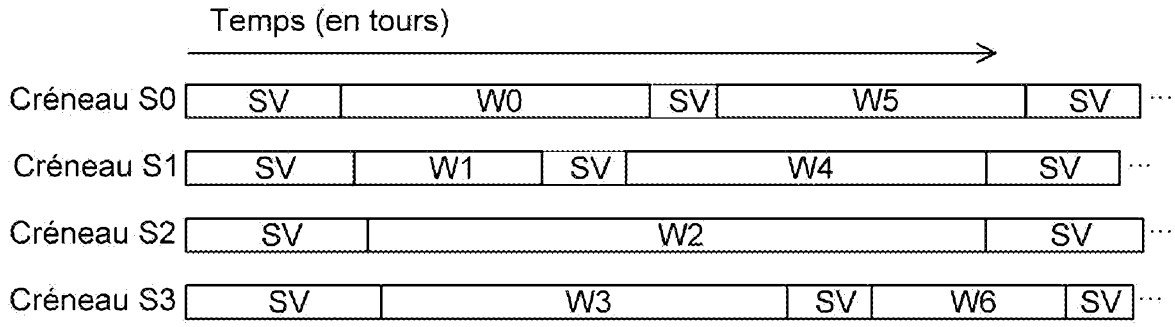
[Fig. 1]



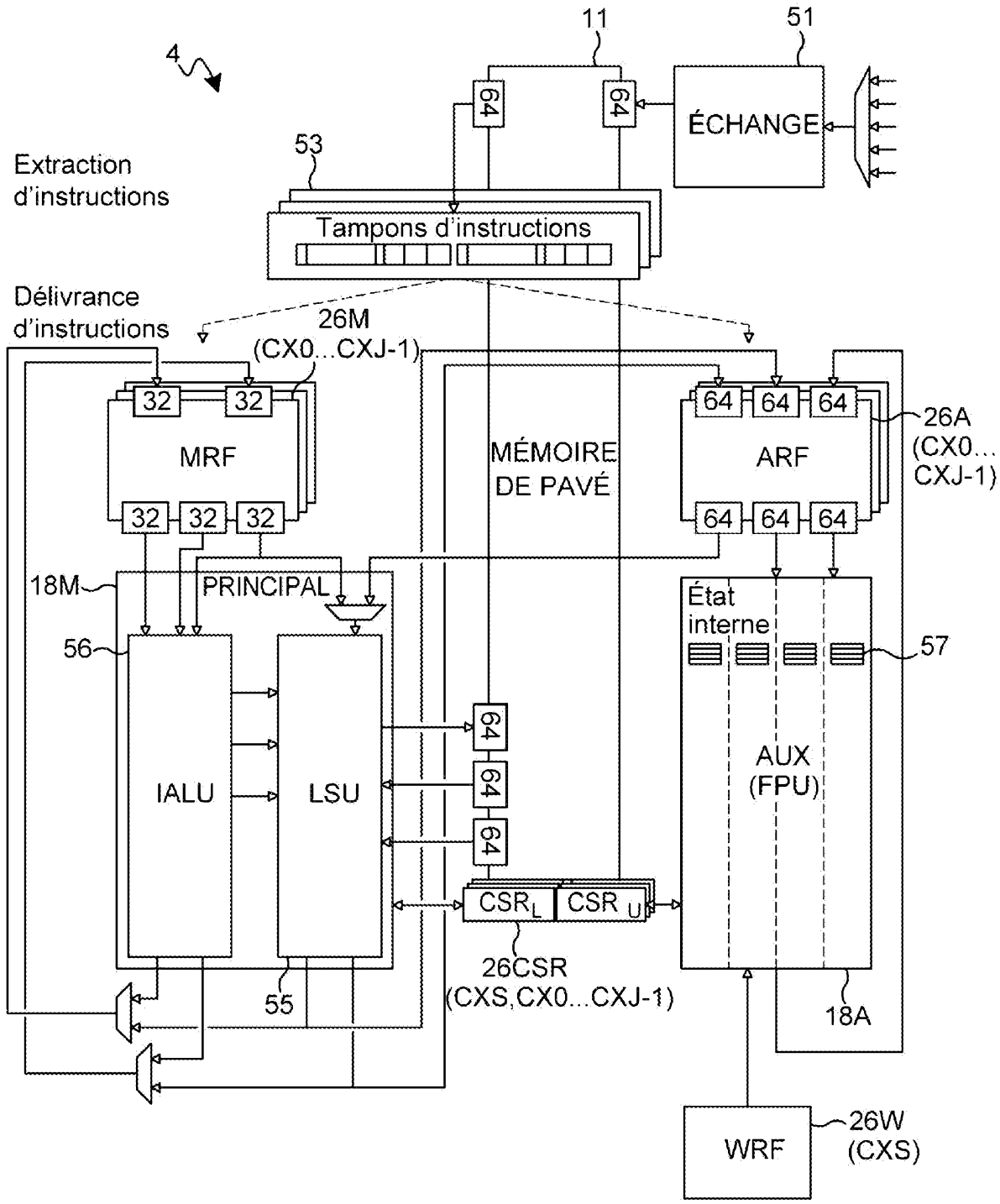
[Fig. 2]



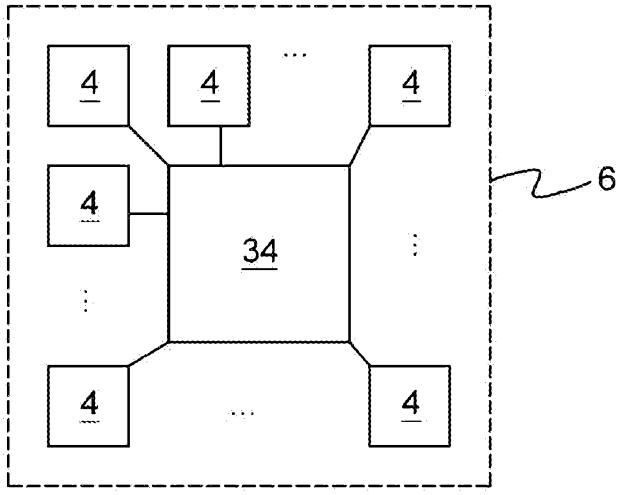
[Fig. 3]



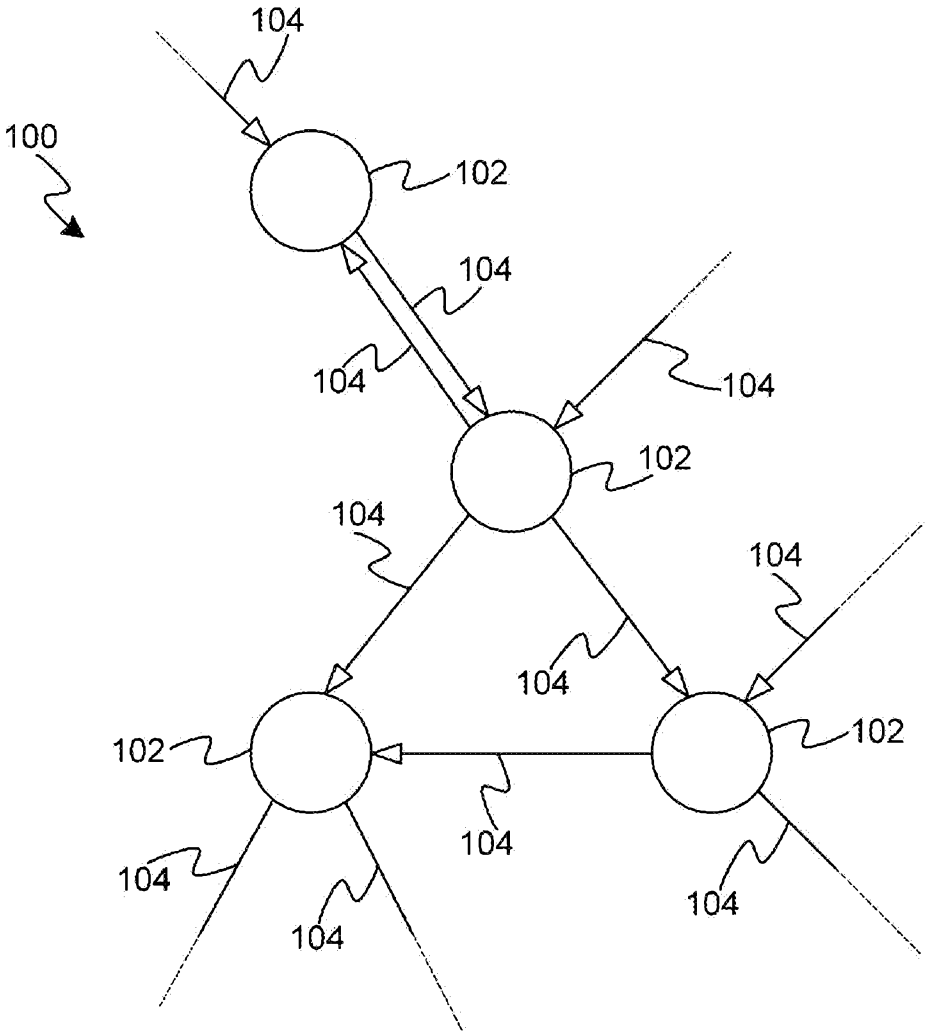
[Fig. 4]


4
11
51
64
64
ÉCHANGE
53
Extraction d'instructions
Tampons d'instructions
Délivrance d'instructions
26M
(CX0...CXJ-1)
32
32
MRF
32
32
32
MÉMOIRE DE PAVÉ
64
64
64
ARF
64
64
64
26A
(CX0... CXJ-1)
18M
PRINCIPAL
56
État interne
57
IALU
LSU
64
64
64
AUX (FPU)
CSR L
CSR U
26CSR
(CXS,CX0...CXJ-1)
55
18A
WRF
26W
(CXS)

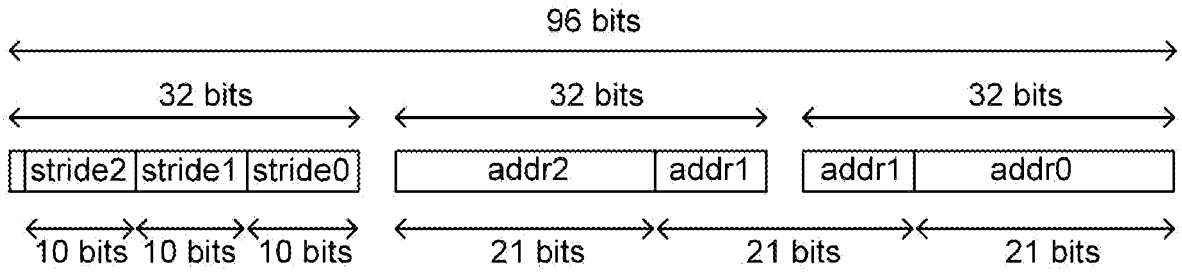
[Fig. 5]


4
4
4
4
34
4
4
6

[Fig. 6]


100
104
102
104
104
104
102
104
104
104
102
102
104
104
104
104

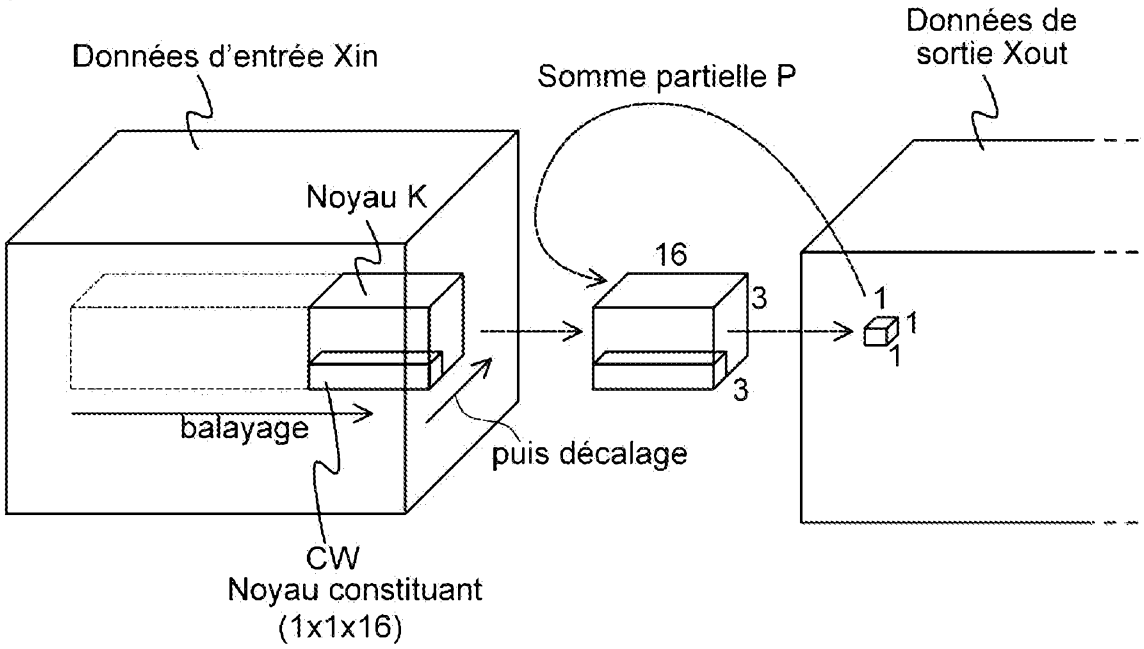
[Fig. 7]



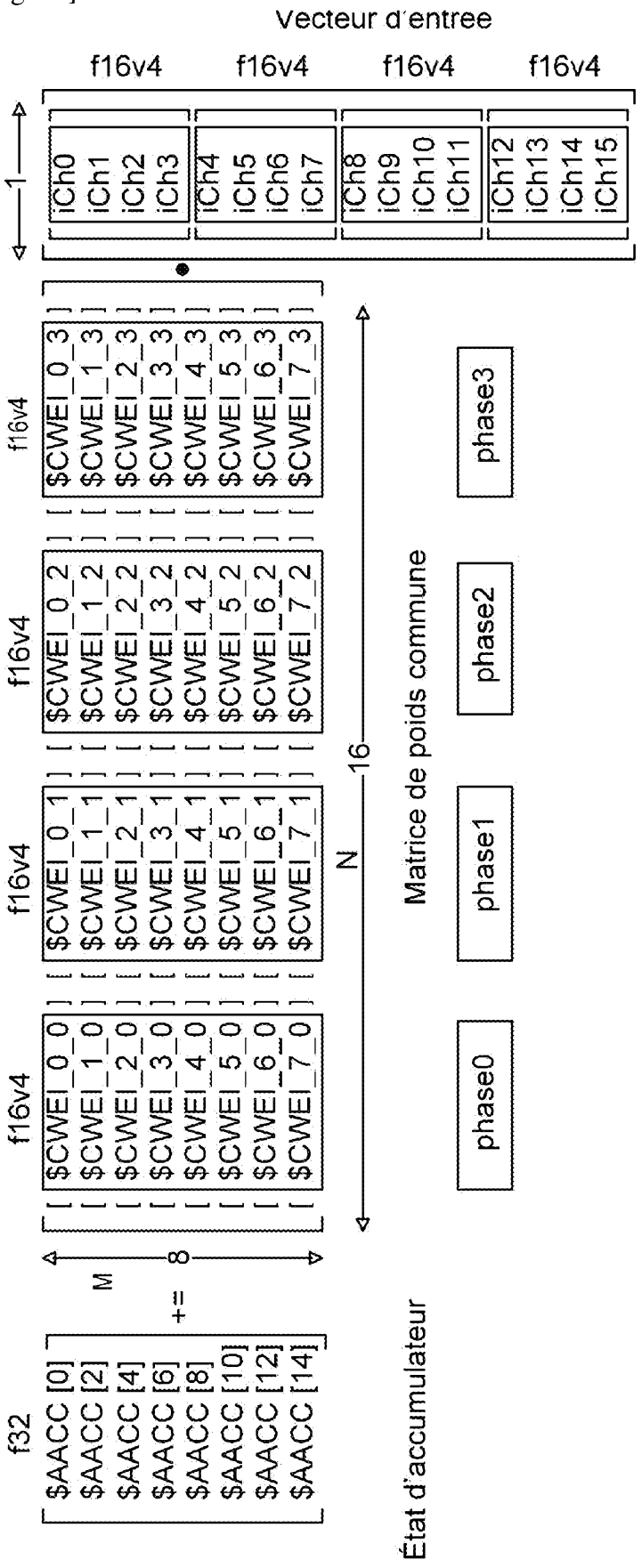
[Fig. 8]

$mStrideA	strideA2	strideA1	strideA0
$mStrideB	strideB2	strideB1	strideB0
$mStrideC	strideC2	strideC1	strideC0
⋮		⋮	

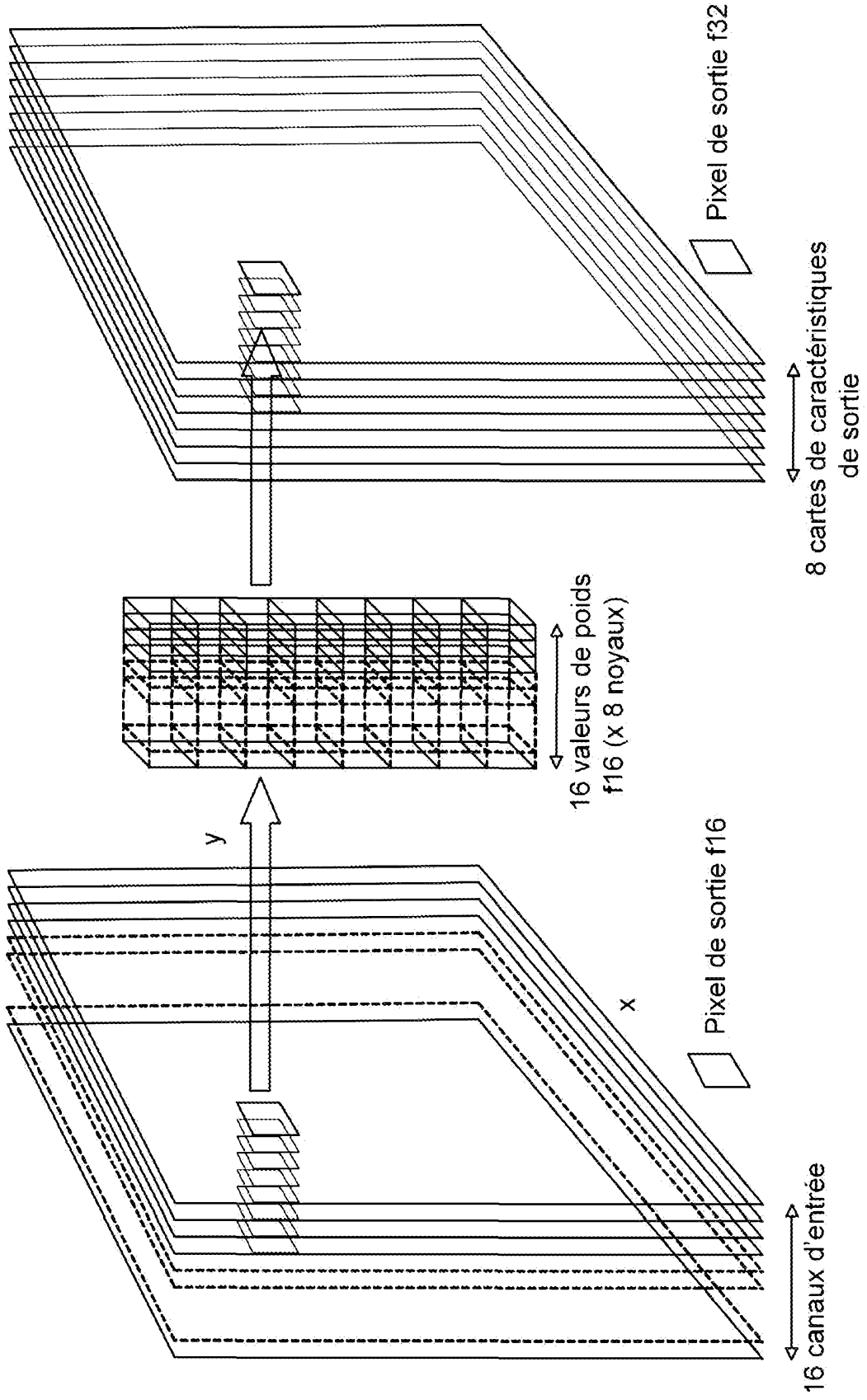
[Fig. 9]



[Fig. 10]


Vecteur d'entree
f16v4
f16v4
f16v4
f16v4
1
iCh0
iCh1
iCh2
iCh3
iCh4
iCh5
iCh6
iCh7
iCh8
iCh9
iCh10
iCh11
iCh12
iCh13
iCh14
iCh15
f16v4
f16v4
f16v4
f16v4
$CWEI_0_3
$CWEI_1_3
$CWEI_2_3
$CWEI_3_3
$CWEI_4_3
$CWEI_5_3
$CWEI_6_3
$CWEI_7_3
$CWEI_0_2
$CWEI_1_2
$CWEI_2_2
$CWEI_3_2
$CWEI_4_2
$CWEI_5_2
$CWEI_6_2
$CWEI_7_2
$CWEI_0_1
$CWEI_1_1
$CWEI_2_1
$CWEI_3_1
$CWEI_4_1
$CWEI_5_1
$CWEI_6_1
$CWEI_7_1
$CWEI_0_0
$CWEI_1_0
$CWEI_2_0
$CWEI_3_0
$CWEI_4_0
$CWEI_5_0
$CWEI_6_0
$CWEI_7_0
N
16
Matrice de poids commune
phase0
phase1
phase2
phase3
M
8
+=
f32
$AACC [0]
$AACC [2]
$AACC [4]
$AACC [6]
$AACC [8]
$AACC [10]
$AACC [12]
$AACC [14]
État d'accumulateur

[Fig. 11]


Pixel de sortie f32
8 cartes de caractéristiques de sortie
16 valeurs de poids f16 (x 8 noyaux)
y
Pixel de sortie f16
x
16 canaux d'entrée

[Fig. 12]

$aSrc0[1	$AACC[14]	$AACC[12]	$AACC[10]	$AACC[8]
$0^3 \| P_0, P_1$	-	-	-	-
$0 \| P_2, P_3$	-	-	-	[MISE EN ACTION
$0 \| P_4, P_5$	-	-	-	-
$0 \| P_6, P_7$	-	-	-	-
$x_0 \| P_8, P_9$	$R_7 = x_0 . CW_{7,0} + P_7$	$R_6 = x_0 . CW_{6,0} + P_6$	$R_5 = x_0 . CW_{5,0} + P_5$	$R_4 = x_0 . CW_{4,0} + P_4$
$x_1 \| P_{10}, P_{11}$	$R_7 = x_1 . CW_{7,1}$	$R_6 = x_1 . CW_{6,1}$	$R_5 = x_1 . CW_{5,1}$	$R_4 = x_1 . CW_{4,1}$
$x_2 \| P_{12}, P_{13}$	$R_7 = x_2 . CW_{7,2}$	$R_6 = x_2 . CW_{6,2}$	$R_5 = x_2 . CW_{5,2}$	$R_4 = x_2 . CW_{4,2}$
$x_3 \| P_{14}, P_{15}$	$R_7 = x_3 . CW_{7,3}$	$R_6 = x_3 . CW_{6,3}$	$R_5 = x_3 . CW_{5,3}$	$R_4 = x_3 . CW_{4,3}$
$x_4 \| P_{16}, P_{17}$	$R_{15} = x_4 . CW_{7,0} + P_{15}$	$R_{14} = x_4 . CW_{6,0} + P_{14}$	$R_{13} = x_4 . CW_{5,0} + P_{13}$	$R_{12} = x_4 . CW_{4,0} + P_{12}$
$x_5 \| P_{18}, P_{19}$	$R_{15} = x_5 . CW_{7,1}$	$R_{14} = x_5 . CW_{6,1}$	$R_{13} = x_5 . CW_{5,1}$	$R_{12} = x_5 . CW_{4,1}$
$x_6 \| P_{20}, P_{21}$	$R_{15} = x_6 . CW_{7,2}$	$R_{14} = x_6 . CW_{6,2}$	$R_{13} = x_6 . CW_{5,2}$	$R_{12} = x_6 . CW_{4,2}$
$x_7 \| P_{22}, P_{23}$	$R_{15} = x_7 . CW_{7,3}$	$R_{14} = x_7 . CW_{6,3}$	$R_{13} = x_7 . CW_{5,3}$	$R_{12} = x_7 . CW_{4,3}$
$x_8 \| P_{24}, P_{25}$	$R_{31} = x_8 . CW_{7,0} + P_{23}$	$R_{22} = x_8 . CW_{6,0} + P_{22}$	$R_{21} = x_8 . CW_{5,0} + P_{21}$	$R_{20} = x_8 . CW_{4,0} + P_{20}$
$x_9 \| P_{26}, P_{27}$	$R_{23} = x_9 . CW_{7,1}$	$R_{22} = x_9 . CW_{6,1}$	$R_{21} = x_9 . CW_{5,1}$	$R_{20} = x_9 . CW_{4,1}$
$x_{10} \| P_{28}, P_{29}$	$R_{23} = x_{10} . CW_{7,2}$	$R_{22} = x_{10} . CW_{6,2}$	$R_{21} = x_{10} . CW_{5,2}$	$R_{20} = x_{10} . CW_{4,2}$
$x_{11} \| P_{30}, P_{31}$	$R_{23} = x_{11} . CW_{7,3}$	$R_{22} = x_{11} . CW_{6,3}$	$R_{21} = x_{11} . CW_{5,3}$	$R_{20} = x_{11} . CW_{4,3}$
$x_{12} \| P_{32}, P_{33}$	$R_{31} = x_{12} . CW_{7,0} + P_{31}$	$R_{30} = x_{12} . CW_{6,0} + P_{30}$	$R_{29} = x_{12} . CW_{5,0} + P_{29}$	$R_{28} = x_{12} . CW_{4,0} + P_{28}$

P_n est une somme partielle d'entrée en simple précision n

x_n est un vecteur d'entrée f16v4

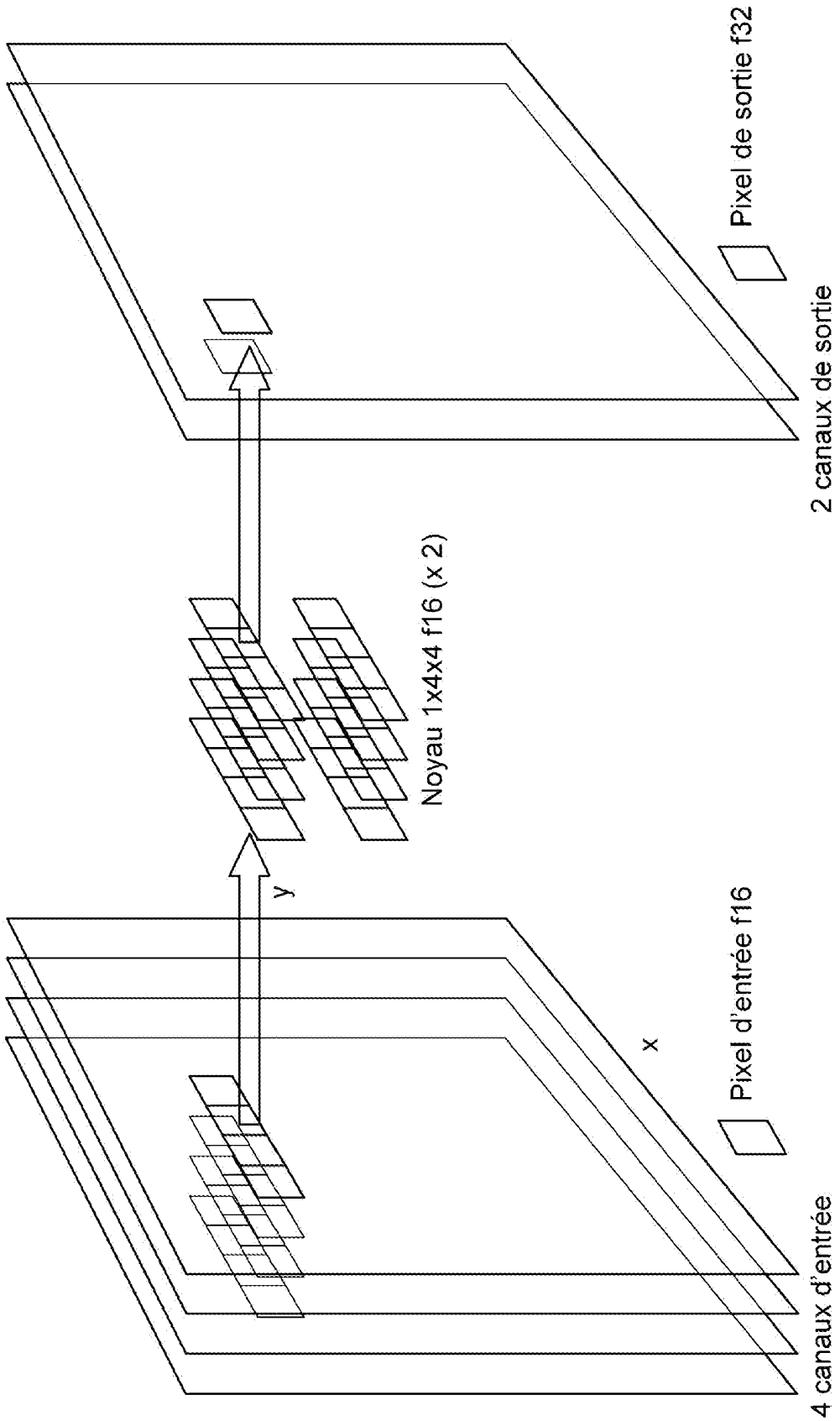
$CW_{m,n}$ est l'état de poids commun $CWEI_m_n

R_n est le résultat final en simple précision d'accumulations de produits scalaires successifs qui ont commencé par P_n

[Fig. 12]

\$AACC[6]	\$AACC[4]	\$AACC[2]	\$AACC[0]	\$aDstO
-	-	-	-	-
PERIODE]	-	-	-	-
-	-	-	-	-
-	-	-	-	-
$R_3=x_0 \cdot CW_{3,0}+P_3$	$R_2=x_0 \cdot CW_{2,0}+P_2$	$R_1=x_0 \cdot CW_{1,0}+P_1$	$R_0=x_0 \cdot CW_{0,0}+P_0$	-
$R_3=x_1 \cdot CW_{3,1}$	$R_2=x_1 \cdot CW_{2,1}$	$R_1=x_1 \cdot CW_{1,1}$	$R_0=x_1 \cdot CW_{0,1}$	-
$R_3=x_2 \cdot CW_{3,2}$	$R_2=x_2 \cdot CW_{2,2}$	$R_1=x_2 \cdot CW_{1,2}$	$R_0=x_2 \cdot CW_{0,2}$	-
$R_3=x_3 \cdot CW_{3,3}$	$R_2=x_3 \cdot CW_{2,3}$	$R_1=x_3 \cdot CW_{1,3}$	$R_0=x_3 \cdot CW_{0,3}$	-
$R_{11}=x_4 \cdot CW_{3,0}+P_{11}$	$R_{10}=x_4 \cdot CW_{2,0}+P_{10}$	$R_9=x_4 \cdot CW_{1,0}+P_9$	$R_8=x_4 \cdot CW_{0,0}+P_8$	R_0,R_1
$R_{11}=x_5 \cdot CW_{3,1}$	$R_{10}=x_5 \cdot CW_{2,1}$	$R_9=x_5 \cdot CW_{1,1}$	$R_8=x_5 \cdot CW_{0,1}$	R_2,R_3
$R_{11}=x_6 \cdot CW_{3,2}$	$R_{10}=x_6 \cdot CW_{2,2}$	$R_9=x_6 \cdot CW_{1,2}$	$R_8=x_6 \cdot CW_{0,2}$	R_4,R_5
$R_{11}=x_7 \cdot CW_{3,3}$	$R_{10}=x_7 \cdot CW_{2,3}$	$R_9=x_7 \cdot CW_{1,3}$	$R_8=x_7 \cdot CW_{0,3}$	R_6,R_7
$R_{19}=x_8 \cdot CW_{3,0}+P_{19}$	$R_{18}=x_8 \cdot CW_{2,0}+P_{18}$	$R_{17}=x_8 \cdot CW_{1,0}+P_{17}$	$R_{16}=x_8 \cdot CW_{0,0}+P_{16}$	R_8,R_9
$R_{19}=x_9 \cdot CW_{3,1}$	$R_{18}=x_9 \cdot CW_{2,1}$	$R_{17}=x_9 \cdot CW_{1,1}$	$R_{16}=x_9 \cdot CW_{0,1}$	R_{10},R_{11}
$R_{19}=x_{10}CW_{3,2}$	$R_{18}=x_{10} \cdot CW_{2,2}$	$R_{17}=x_{10} \cdot CW_{1,2}$	$R_{16}=x_{10} \cdot CW_{0,2}$	R_{12},R_{13}
$R_{19}=x_{11} \cdot CW_{3,3}$	$R_{18}=x_{11} \cdot CW_{2,3}$	$R_{17}=x_{11} \cdot CW_{1,3}$	$R_{16}=x_{11} \cdot CW_{0,3}$	R_{14},R_{15}
$R_{27}=x_{12} \cdot CW_{3,0}+P_{27}$	$R_{26}=x_{12} \cdot CW_{2,0}+P_{26}$	$R_{25}=x_{12} \cdot CW_{1,0}+P_{25}$	$R_{24}=x_{12}CW_{0,0}+P_{24}$	R_{16},R_{17}

[Fig. 13]


Pixel de sortie f32
2 canaux de sortie
Noyau 1x4x4 f16 (x 2)
y
Pixel d'entrée f16
x
4 canaux d'entrée

[Fig. 14]

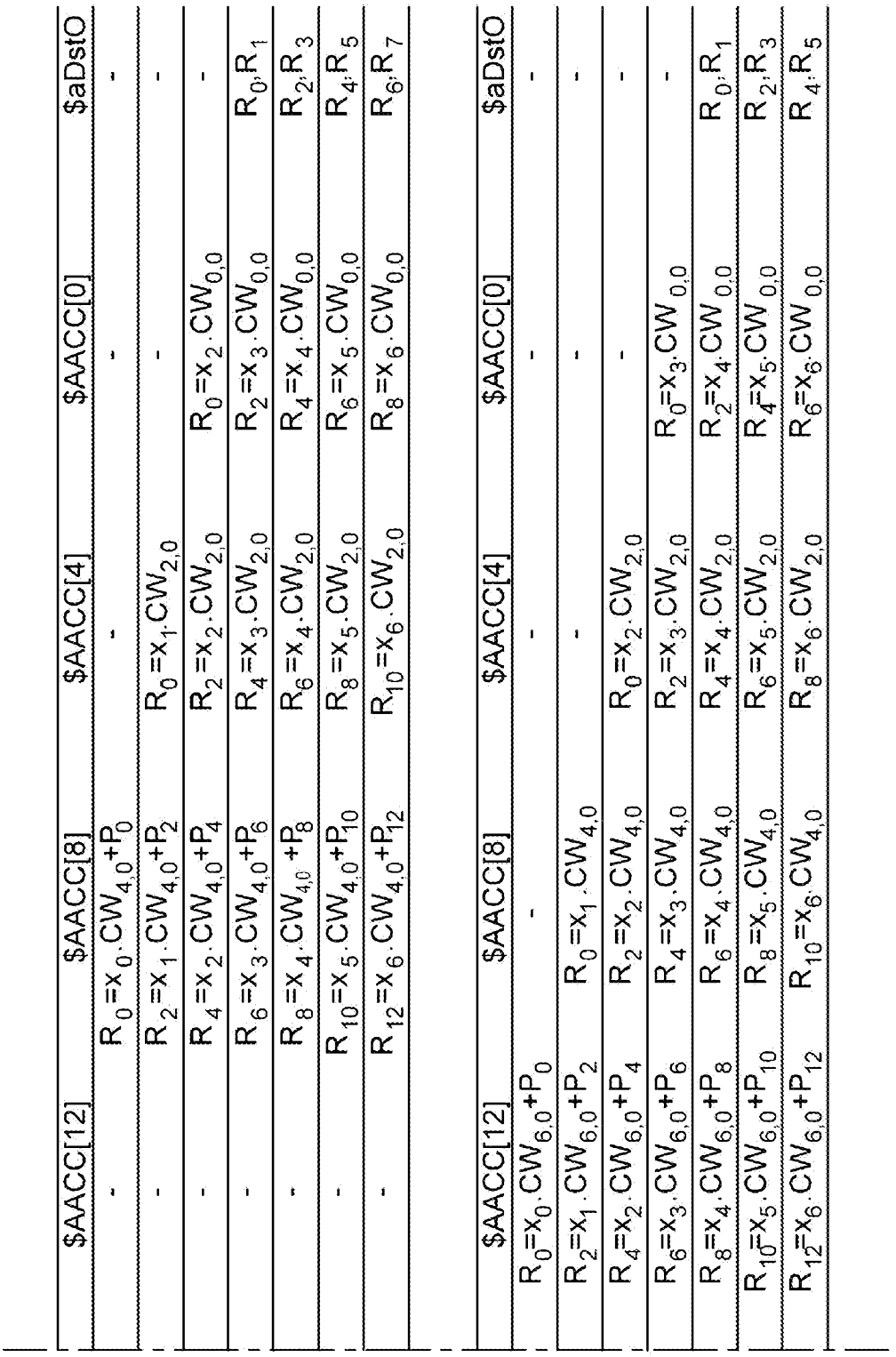
$aSrc0|1	$AACC[14]	$AACC[10]	$AACC[6]	$AACC[2]
$x_0\|P_0,P_1$	-	$R_1=x_0.CW_{5,0}+P_1$	-	-
$x_1\|P_2,P_3$	-	$R_3=x_1.CW_{5,0}+P_3$	$R_1=x_1.CW_{3,0}$	-
$x_2\|P_4,P_5$	-	$R_5=x_2.CW_{5,0}+P_5$	$R_3=x_2.CW_{3,0}$	$R_1=x_2.CW_{1,0}$
$x_3\|P_6,P_7$	-	$R_7=x_3.CW_{5,0}+P_7$	$R_5=x_3.CW_{3,0}$	$R_3=x_3.CW_{1,0}$
$x_4\|P_8,P_9$	-	$R_9=x_4.CW_{5,0}+P_9$	$R_7=x_4.CW_{3,0}$	$R_5=x_4.CW_{1,0}$
$x_5\|P_{10},P_{11}$	-	$R_{11}=x_5.CW_{5,0}+P_{11}$	$R_9=x_5.CW_{3,0}$	$R_7=x_5.CW_{1,0}$
$x_6\|P_{12},P_{13}$	-	$R_{13}=x_6.CW_{5,0}+P_{13}$	$R_{11}=x_6.CW_{3,0}$	$R_9=x_6.CW_{1,0}$

Tableau 3.156 : f16v4sisoslic, exemple de séquence 1x4

$aSrc0|1	$AACC[14]	$AACC[10]	$AACC[6]	$AACC[2]
$x_0\|P_0,P_1$	$R_1=x_0.CW_{7,0}+P_1$	-	-	-
$x_1\|P_2,P_3$	$R_3=x_1.CW_{7,0}+P_3$	$R_1=x_1.CW_{5,0}$	-	-
$x_2\|P_4,P_5$	$R_5=x_2.CW_{7,0}+P_5$	$R_3=x_2.CW_{5,0}$	$R_1=x_2.CW_{3,0}$	-
$x_3\|P_6,P_7$	$R_7=x_3.CW_{7,0}+P_7$	$R_5=x_3.CW_{5,0}$	$R_3=x_3.CW_{3,0}$	$R_1=x_3.CW_{1,0}$
$x_4\|P_8,P_9$	$R_9=x_4.CW_{7,0}+P_9$	$R_7=x_4.CW_{5,0}$	$R_5=x_4.CW_{3,0}$	$R_3=x_4.CW_{1,0}$
$x_5\|P_{10},P_{11}$	$R_{11}=x_5.CW_{7,0}+P_{11}$	$R_9=x_5.CW_{5,0}$	$R_7=x_5.CW_{3,0}$	$R_5=x_5.CW_{1,0}$
$x_6\|P_{12},P_{13}$	$R_{13}=x_6.CW_{7,0}+P_{13}$	$R_{11}=x_6.CW_{5,0}$	$R_9=x_6.CW_{3,0}$	$R_7=x_6.CW_{1,0}$

P_n est une somme partielle d'entrée en simple précision n
x_n est un vecteur d'entrée f16v4
$CW_{m,n}$ est l'état de poids commun $CWEI_m_n
R_n est le résultat final en simple précision d'accumulations de produits scalaires successifs qui ont commencé par P_n

[Fig. 14]


$AACC[12] $AACC[8] $AACC[4] $AACC[0] $aDstO
- $R_0=x_0.CW_{4,0}+P_0$ - - -
- $R_2=x_1.CW_{4,0}+P_2$ $R_0=x_1.CW_{2,0}$ - -
- $R_4=x_2.CW_{4,0}+P_4$ $R_2=x_2.CW_{2,0}$ $R_0=x_2.CW_{0,0}$ -
- $R_6=x_3.CW_{4,0}+P_6$ $R_4=x_3.CW_{2,0}$ $R_2=x_3.CW_{0,0}$ R_0,R_1
- $R_8=x_4.CW_{4,0}+P_8$ $R_6=x_4.CW_{2,0}$ $R_4=x_4.CW_{0,0}$ R_2,R_3
- $R_{10}=x_5.CW_{4,0}+P_{10}$ $R_8=x_5.CW_{2,0}$ $R_6=x_5.CW_{0,0}$ R_4,R_5
- $R_{12}=x_6.CW_{4,0}+P_{12}$ $R_{10}=x_6.CW_{2,0}$ $R_8=x_6.CW_{0,0}$ R_6,R_7
$AACC[12] $AACC[8] $AACC[4] $AACC[0] $aDstO
$R_0=x_0.CW_{6,0}+P_0$ - - - -
$R_2=x_1.CW_{6,0}+P_2$ $R_0=x_1.CW_{4,0}$ - - -
$R_4=x_2.CW_{6,0}+P_4$ $R_2=x_2.CW_{4,0}$ $R_0=x_2.CW_{2,0}$ - -
$R_6=x_3.CW_{6,0}+P_6$ $R_4=x_3.CW_{4,0}$ $R_2=x_3.CW_{2,0}$ $R_0=x_3.CW_{0,0}$ -
$R_8=x_4.CW_{6,0}+P_8$ $R_6=x_4.CW_{4,0}$ $R_4=x_4.CW_{2,0}$ $R_2=x_4.CW_{0,0}$ R_0,R_1
$R_{10}=x_5.CW_{6,0}+P_{10}$ $R_8=x_5.CW_{4,0}$ $R_6=x_5.CW_{2,0}$ $R_4=x_5.CW_{0,0}$ R_2,R_3
$R_{12}=x_6.CW_{6,0}+P_{12}$ $R_{10}=x_6.CW_{4,0}$ $R_8=x_6.CW_{2,0}$ $R_6=x_6.CW_{0,0}$ R_4,R_5