(19) RÉPUBLIQUE FRANÇAISE

INSTITUT NATIONAL
DE LA PROPRIÉTÉ INDUSTRIELLE

COURBEVOIE

(11) **N° de publication :** **3 019 919**
(à n'utiliser que pour les commandes de reproduction)

(21) **N° d'enregistrement national :** **14 53308**

(51) Int Cl[8] : ***G 06 F 9/46*** *(2013.01)*

(12) DEMANDE DE BREVET D'INVENTION A1

(22) **Date de dépôt :** 14.04.14.

(30) **Priorité :**

(43) **Date de mise à la disposition du public de la demande :** 16.10.15 Bulletin 15/42.

(56) **Liste des documents cités dans le rapport de recherche préliminaire :** *Se reporter à la fin du présent fascicule*

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

( ) **Demande(s) d'extension :**

(71) **Demandeur(s) :** *INRIA INSTITUT NATIONAL DE RECHERCHE EN INFORMATIQUE ET EN AUTOMATIQUE Etablissement public* — FR.

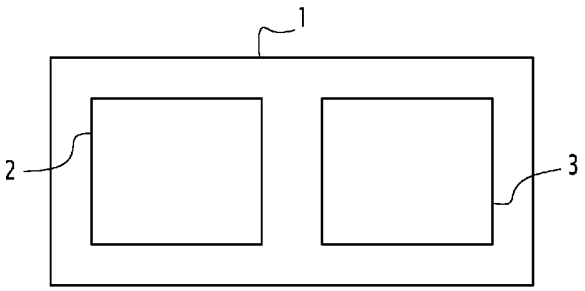
(72) **Inventeur(s) :** ALIAS CHRISTOPHE et PLESCO ALEXANDRU.

(73) **Titulaire(s) :** INRIA INSTITUT NATIONAL DE RECHERCHE EN INFORMATIQUE ET EN AUTOMATIQUE Etablissement public.

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

(54) PROCEDE DE SYNTHESE AUTOMATIQUE DE CIRCUITS, DISPOSITIF ET PROGRAMME D'ORDINATEUR ASSOCIES.

(57) Procédé de synthèse automatique de circuits comprenant la génération d'un réseau de processus réguliers lisant ou écrivant des données dans des canaux, selon lequel un unique processus producteur est autorisé à écrire dans un canal et un unique processus consommateur est autorisé à lire dans un canal ; et une unité de synchronisation associée audit canal autorise ou gèle la mise en oeuvre d'une nouvelle itération dudit processus producteur, respectivement consommateur, en fonction d'une comparaison d'une position d'exécution déterminée en fonction de la valeur d'une nouvelle itération collectée du processus producteur, respectivement consommateur, et d'une position d'exécution déterminée en fonction d'une dernière valeur d'itération collectée du processus consommateur, respectivement producteur.



FR 3 019 919 - A1

Procédé de synthèse automatique de circuits, dispositif et programme d'ordinateur associés

La présente invention concerne un procédé de synthèse automatique de circuits comprenant une étape de compilation d'un programme logiciel en un réseau de processus réguliers interconnectés à l'aide de canaux, selon lequel chaque canal est une mémoire physique adressable, chacun desdits processus applique, lors de chaque itération d'un domaine d'itérations qui lui est associé, au moins une lecture d'une donnée dans un canal et/ou une écriture d'une donnée dans un canal, et une fonction d'ordonnancement fournit, pour chaque valeur d'itération de chaque processus, une position d'exécution du processus sélectionnée dans un ensemble temporellement ordonné de positions d'exécution.

De tels procédés de synthèse de circuits, ou HLS, (en anglais « high-level synthesis »), sont utilisés pour générer automatiquement (ou compiler) un bloc de circuits électroniques à partir d'une description algorithmique (haut niveau telle que le langage C ou Matlab, Alpha) du calcul à effectuer par le circuit (cf. des outils tels que SpecC, HardwareC, HandelC, BashC, TransmogrifierC, CyberC, CatapultC). Les algorithmes concernés sont divers, par exemple des algorithmes de traitement du signal.

Le circuit doit être le plus efficace possible tout en utilisant au mieux les ressources disponibles. Les performances des blocs de circuits générés sont ainsi mesurées notamment par la vitesse de traitement des données fournies en entrée, par la consommation énergétique, la surface de silicium occupée, unités LUT sur FPGA, accès mémoire etc.

Une augmentation des performances est obtenue en divisant le traitement à effectuer en tâches à réaliser en parallèle. La plupart du temps, les tâches ne sont pas totalement indépendantes, et doivent communiquer (écrire/lire) des résultats intermédiaires entre elles, ce qui pose notamment le problème de la synchronisation des tâches entre elles.

Le découpage en tâches parallèles et la production des synchronisations nécessitent un ordonnancement à grain fin du calcul, c'est-à-dire une réorganisation de l'ordre d'exécution des opérations de l'algorithme. Pour être correct, un ordonnancement doit respecter les dépendances de données entre opérations, pour assurer, par exemple, qu'une donnée est produite avant d'être lue.

La plupart des approches existantes s'appuient sur une représentation de type CDFG (Control/Data Flow Graph), qui expriment des dépendances entre constructions

syntaxiques, c'est-à-dire entre paquets d'opérations, ce qui mène à une sur-approximation grossière des dépendances telle qu'un ordonnancement intéressant ne peut être trouvé.

Les réseaux de processus sont des modèles d'exécution qui expriment naturellement le parallélisme d'un calcul. A ce titre, ils sont utilisés pour représenter des systèmes parallèles, et ces traductions de systèmes parallèles en réseaux de processus permettent d'étudier, analyser et mesurer différentes problématiques liées au parallélisme (ordonnancement, équilibrage de charge, allocation, checkpointing, consommation, etc).

Les réseaux de processus constituent en outre une représentation intermédiaire naturelle pour un dispositif de synthèse automatique de circuits, dans lequel un bloc « Front-End » extrait le parallélisme et produit un réseau de processus, et un bloc « Back-End » compile le réseau de processus vers l'architecture cible.

Si de nombreuses avancées en termes de performances ont été réalisées sur les aspects back-end (construction des pipelines, placement/routage), les aspects front-end restent encore rudimentaires.

Il est ainsi souhaitable d'augmenter encore les performances des blocs de circuits générés automatiquement.

A cet effet, suivant un premier aspect, l'invention propose un procédé de synthèse automatique de circuits du type précité caractérisé en ce qu'il comprend les étapes suivantes, lors de l'étape de compilation, pour chacun desdits canaux :

un unique processus, dit processus producteur, est autorisé à écrire des données dans le canal et un unique processus, dit processus consommateur, est autorisé à lire des données dans ledit canal ; et une unité de synchronisation est associée audit canal, et telle qu'elle collecte, préalablement au lancement de chaque nouvelle itération par le processus producteur du canal, la valeur de ladite nouvelle itération du processus producteur, et collecte, préalablement au lancement de chaque nouvelle itération par le processus consommateur du canal, la valeur d'itération du processus consommateur ; ladite unité de synchronisation, suite à la collecte de la valeur d'une nouvelle itération du processus producteur, respectivement consommateur, autorisant ou gelant la mise en œuvre de ladite nouvelle itération dudit processus producteur, respectivement consommateur, en fonction d'une comparaison d'une position d'exécution déterminée en fonction de la valeur de nouvelle itération collectée du processus producteur, respectivement consommateur, et d'une position d'exécution déterminée en fonction de la dernière valeur d'itération collectée du processus consommateur , respectivement producteur.

Un tel procédé permet de générer rapidement et de manière automatique des circuits offrant de grandes performances en terme de vitesse de calcul et de volume occupé par le circuit.

Dans des modes de réalisation, le procédé de synthèse automatique de circuit suivant l'invention comporte en outre une ou plusieurs des caractéristiques suivantes :

- ladite unité de synchronisation est telle qu'elle pilote le fonctionnement du processus producteur dudit canal en mettant en œuvre les étapes suivantes suite à la collecte de la valeur d'une nouvelle itération dudit processus producteur :
    - l'unité de synchronisation compare la nouvelle position d'exécution de ladite valeur collectée, à la position d'exécution de la dernière valeur d'itération du processus consommateur collectée ; et
    - ladite unité de synchronisation gèle la mise en œuvre par le processus producteur de cette nouvelle itération si la position d'exécution de cette nouvelle itération est postérieure à la position d'exécution de la dernière valeur d'itération collectée du processus consommateur, le gel étant maintenu par l'unité de synchronisation jusqu'à la collecte d'une future valeur d'itération du processus consommateur telle que la position d'exécution de cette future valeur d'itération soit postérieure à ladite position d'exécution de la nouvelle itération du processus consommateur ;
- ladite unité de synchronisation est telle qu'elle pilote le fonctionnement du processus consommateur dudit canal en mettant en œuvre les étapes suivantes suite à la collecte de la valeur d'une nouvelle itération dudit processus consommateur :
    - l'unité de synchronisation détermine la position d'exécution de celle des valeurs d'itération du processus producteur qui donne lieu à l'écriture dans ledit canal (c) de la donnée qui doit être lue par le processus consommateur lors de cette nouvelle itération ; et
    - ladite unité de synchronisation gèle la mise en œuvre par le processus consommateur de sa nouvelle itération si la position d'exécution déterminée est postérieure ou égale à la position d'exécution de la dernière valeur d'itération collectée du processus producteur, le gel étant maintenu par l'unité de synchronisation jusqu'à la collecte d'une future valeur d'itération du processus producteur telle que la position d'exécution de cette future valeur d'itération soit supérieure strictement à ladite position d'exécution déterminée.
- le processus consommateur ou producteur de chaque canal après la fin d'une précédente itération dudit processus et préalablement au lancement de la nouvelle

itération succédant à la précédente itération, délivre à l'unité de synchronisation du canal, la valeur d'itération de la nouvelle itération du processus, et met en œuvre cette nouvelle itération du processus en fonction d'une commande de pilotage de l'unité de supervision indiquant un gel ou une autorisation ;

- le processus consommateur du canal après la fin d'une précédente itération dudit processus et préalablement au lancement de la nouvelle itération succédant à la précédente itération, délivre en outre à l'unité de synchronisation du canal, ladite valeur d'itération du processus producteur qui donne lieu à l'écriture dans ledit canal de la donnée qui doit être lue par le processus consommateur lors de cette nouvelle itération.

Suivant un deuxième aspect, la présente invention propose un dispositif de synthèse automatique de circuits adapté pour effectuer une compilation d'un programme logiciel en générant un réseau de processus réguliers interconnectés à l'aide de canaux, dans lequel chaque canal est une mémoire physique adressable, chacun desdits processus étant adapté pour appliquer, lors de chaque itération d'un domaine d'itérations qui lui est associé, au moins une lecture d'une donnée dans un canal et/ou une écriture d'une donnée dans un canal, et une fonction d'ordonnancement étant adaptée pour fournir, pour chaque valeur d'itération de chaque processus, une position d'exécution du processus sélectionnée dans un ensemble temporellement ordonné de positions d'exécution ; ledit dispositif de synthèse de circuits étant caractérisé en ce qu'il est adapté pour lors de ladite compilation, générer et associer à chacun desdits canaux :

un unique processus, dit processus producteur, autorisé à écrire des données dans le canal et un unique processus, dit processus consommateur, autorisé à lire des données dans ledit canal ; et une unité de synchronisation associée audit canal et adaptée pour collecter, préalablement au lancement de chaque nouvelle itération par le processus producteur du canal, la valeur de ladite nouvelle itération du processus producteur, et collecter, préalablement au lancement de chaque nouvelle itération par le processus consommateur du canal, la valeur d'itération du processus consommateur ; ladite unité de synchronisation, suite à la collecte de la valeur d'une nouvelle itération du processus producteur, respectivement consommateur, étant adaptée pour autoriser ou geler la mise en œuvre de ladite nouvelle itération dudit processus producteur, respectivement consommateur, en fonction d'une comparaison d'une position d'exécution déterminée en fonction de la valeur de nouvelle itération collectée du processus producteur, respectivement consommateur, et d'une position d'exécution déterminée en fonction de la dernière valeur d'itération collectée du processus consommateur, respectivement producteur.

Suivant un troisième aspect, la présente invention propose un circuit synthétisé par exemple suite à la mise en œuvre d'un procédé de synthèse automatique de circuit selon le premier aspect de l'invention, comprenant un réseau de processus réguliers interconnectés à l'aide de canaux, dans lequel chaque canal est une mémoire physique adressable, chacun desdits processus étant adapté pour appliquer, lors de chaque itération d'un domaine d'itérations qui lui est associé, au moins une lecture d'une donnée dans un canal et/ou une écriture d'une donnée dans un canal, et une fonction d'ordonnancement étant adaptée pour fournir, pour chaque valeur d'itération de chaque processus, une position d'exécution du processus sélectionnée dans un ensemble temporellement ordonné de positions d'exécution ;

ledit circuit étant caractérisé en ce qu'il comprend :

un unique processus, dit processus producteur, associé à chaque canal et autorisé à écrire des données dans le canal et un unique processus, dit processus consommateur, associé à chaque canal et autorisé à lire des données dans ledit canal ; et

une unité de synchronisation associée audit canal et adaptée pour collecter, préalablement au lancement de chaque nouvelle itération par le processus producteur du canal, la valeur de ladite nouvelle itération du processus producteur, et collecter, préalablement au lancement de chaque nouvelle itération par le processus consommateur du canal, la valeur d'itération du processus consommateur ; ladite unité de synchronisation, suite à la collecte de la valeur d'une nouvelle itération du processus producteur, respectivement consommateur, étant adaptée pour autoriser ou geler la mise en œuvre de ladite nouvelle itération dudit processus producteur, respectivement consommateur, en fonction d'une comparaison d'une position d'exécution déterminée en fonction de la valeur de nouvelle itération collectée du processus producteur, respectivement consommateur, et d'une position d'exécution déterminée en fonction de la dernière valeur d'itération collectée du processus consommateur, respectivement producteur.

Ces caractéristiques et avantages de l'invention apparaîtront à la lecture de la description qui va suivre, donnée uniquement à titre d'exemple, et faite en référence aux dessins annexés, sur lesquels :

- la figure 1 représente un dispositif de synthèse automatique de circuits dans un mode de réalisation de l'invention ;
- la figure 2 illustre les domaines d'itération et de dépendances du programme nommé Prog ;
- la figure 3 illustre l'ordre d'exécution du programme Prog ;

- la figure 4 est une vue de processus, multiplexeurs d'entrée, démultiplexeurs de sortie, canaux mis en œuvre dans la génération automatique du bloc de circuits exécutant la programme Prog ;
- la figure 5 est une vue représentant deux processus du programme Prog, du canal entre ces deux processus et de l'unité de synchronisation associée au canal.

La figure 1 est une vue d'un dispositif 1 de synthèse automatique de circuits haut-niveau dans un mode de réalisation de l'invention. Ce dispositif 1 comporte un bloc 2, dit « Front-End » et un bloc 3, dit « Back-End ».

Le bloc « Front-End » 2 analyse la description algorithmique d'un programme fourni en entrée du dispositif 1 pour en extraire le parallélisme et il produit un réseau de processus en tant que représentation intermédiaire du circuit correspondant.

Le bloc « Back-End » 3 transforme la représentation intermédiaire du circuit produite par le bloc « Front-End » 2 en circuit physique.

Ci-après des notions et traitements utilisés dans les dispositifs de synthèse automatique de circuits haut niveau, et également dans le bloc « Front-End » 2 selon l'invention sont tout d'abord rappelés.

Notions et traitements

Modèle polyédrique

Le modèle polyédrique est un outil d'analyses et d'optimisations des programmes d'ordinateur à contrôle statique. Les programmes à contrôle statique ne manipulent que des tableaux statiques et des variables de type simple (pas de pointeurs) et dont le contrôle est limité aux boucles « for » et « if ». Une caractéristique essentielle est que les fonctions d'indice des tableaux, les bornes des boucles et les tests des « if » sont des fonctions affines des compteurs des boucles « for » englobantes.

Le vecteur d'itération $\vec{i}$ d'une affectation S est formé des compteurs des boucles qui l'englobent. L'instance d'exécution (S, $\vec{i}$) est appelée opération. Les opérations sont les unités de base des analyses polyédriques. Le domaine d'itération d'une affectation est l'ensemble de tous ses vecteurs d'itération au cours de l'exécution. Sous ces restrictions (boucles « for », « if », contraintes affines), le domaine d'itération d'une affectation est exactement un polyèdre convexe, dont la forme ne dépend pas des entrées de l'affectation. On peut alors le calculer statiquement, et appliquer diverses opérations

polyédriques (opérations géométriques, optimisation linéaire, comptage, etc.) pour construire des analyses statiques et transformations de programmes.

Ci-après, on distinguera dans chaque opération W = f (R_1, ..., R_n) les lectures (R_1, ..., R_n) et écritures (W). Pour un programme à contrôle statique P, on note Ω(P) l'ensemble des opérations élémentaires ainsi obtenu. On note p_{seq} l'ordre d'exécution séquentiel sur Ω(P). On convient que les lectures R_i d'une opération sont exécutées avant l'écriture : $R_i \prec_{seq} W, \forall i$. Enfin, on note $\mathrm{M}(P)$ l'ensemble des emplacements mémoire accédés par chaque opération élémentaire X E Ω(P). On note μ(X) l'emplacement accédé (en lecture ou en écriture) par X, μ : Ω(P) —> $\mathrm{M}(P)$.

Par commodité, dans toute la suite, on appellera opération une opération élémentaire.

Dépendances, ordonnancement

Il existe une dépendance X —> Y entre deux opérations X, Y∈ Ω (P) lorsque X s'exécute avant $Y(X \prec_{seq} Y)$ et lorsque X et Y accèdent au même emplacement mémoire (μ(X) = μ(Y)). Selon la nature de X et de Y (lecture ou écriture), on parle de dépendance de flot (W puis R), anti (R puis W) ou sortie (W puis W). Les dépendances anti et sortie n'expriment que des conflits d'accès à la mémoire et peuvent être supprimés en modifiant l'utilisation de la mémoire En réalité, seules les dépendances de flot comptent, qui expriment le calcul réalisé par le programme. Plus précisément, on s'intéresse aux dépendances de flot W —> R ou W est la dernière écriture de μ(R) qui précède R, autrement dit le W qui définit la valeur lue par R. Ce W, qui est unique, est appelé source de R. On le note s(R).

On a : $s(R) = \max_{\prec seq} \{W \in \Omega(P), \mu(W) = \mu(R) \wedge W \prec_{seq} R\}$, l'opérateur $\wedge$ signifiant « et ».

Un ordonnancement est une application θ qui associe à chaque opération du programme une date d'exécution, qui appartient à un ensemble totalement ordonné, θ : Ω (P) —> (T, <<).

La plupart des transformations de programmes (optimisations etc.) peuvent s'écrire avec une fonction d'ordonnancement. Le modèle polyédrique fournit des techniques pour calculer des ordonnancements affines $\theta(S, \vec{i}) = A\vec{i} + \vec{b} \in (\mathrm{N}^P, <<)$, où << désigne l'ordre lexicographique. Un ordonnancement est correct s'il respecte la causalité

des dépendances : i.e s'il existe la dépendance X —> Y entre deux opérations X, Y $\Rightarrow$ $\theta(X) << \theta(Y)$, i.e. l'ordre d'exécution de X est antérieur à celui de Y. L'ordre d'exécution induit par θ est noté $\prec_\theta$. On définit s_θ en remplaçant mutatis mutandis $\prec_{seq}$ par $\prec_\theta$ dans la relation relative à s(R) indiquée ci-dessus.

A titre d'exemple, considérons le programme Prog suivant exprimé en langage de haut niveau :

```
I1 : a[0] = 0;
I2 : a[N-1] = 0;
for(t=0; t ≤ K-1; t++)
 {
  for(i=1; i ≤  N-2; i++)
S :   b[i] = a[i-1] + a[i] + a[i+1];
  for(i=1; i ≤  N-2; i++)
T :   a[i] = b[i];
 }
R : result = a[1];
```

Ce programme Prog est un programme Jacobi 1D à contrôle statique qui effectue un calcul de relaxation binomiale itératif. Les bords du tableau a sont initialisés (affectations I1 et I2), ensuite, on calcule itérativement une moyenne sur chaque fenêtre de trois cases (affectations S et T). Enfin, on récupère le résultat (affectation R).

Les domaines d'itération des affectations sont donnés en représentés en figure 2. Les domaines des affectations I_1, I_2 et R sont réduits à un point. Les domaines des affectations S et T sont des rectangles, qu'on superpose ici pour faciliter la présentation. Les points noirs représentent les itérations de S, et les croix représentent les itérations de T.

Ce programme comporte tous les types de dépendances, dont quelques instances sont représentées par des flèches sur le dessin.

- Dépendances de flot : (I_1,) —> (S, t,), (I_2,) —> (S, t, N - 2), (S, t, i) —> (T, t, i), (T, t, i) —> (S, t + 1, i - 1), (T, t, i) —> (S, t + 1, i), (T, t, i) —> (S, t + 1, i + 1), (T, K - 1,1) —> (R,).
- Dépendances anti (en pointillés) : (S, t, i) —> (T, t, i-1), (S, t, i) —> (T, t, i), (S, t, i) —>
(T,t,i+1), (T, t, i) —> (S,t+1,i)
- Dépendances de sortie : (S, t, i) —> (S, t+1, i) (T, T-1,1) —> (R,), toutes ces dépendances sont incluses dans les dépendances de flot, elles ne sont donc pas représentées.

Un ordonnancement valide est θ(I1,) = θ(I2,) = (0), θ(S, t, i) = (1, 2t+i, t, 0), θ(T, t, i) = (1, 2t + i +1, t,1), θ (R,) = (2).

L'ordre d'exécution correspondant est le suivant. On exécute d'abord I1 et I2 en même temps. Ensuite, on exécute les instances de S et T de façon entrelacée, comme indiqué sur le dessin. Enfin, on exécute R.

Cet ordonnancement est valide, en ce sens qu'il satisfait bien toutes les dépendances. Par exemple, la première dépendance anti est respectée puisque θ(S,t,i)=(1,2t+i,t,0)<<θ(T,t,i-1)=(1,2t+i-1+1, t, 1) = (1,2t + i, t, 1).

Allocation mémoire, assignation unique

Une allocation mémoire est une application σ qui associe à chaque emplacement mémoire m ∈ M(P) adressé par P, un emplacement alternatif σ(m) dans un nouvel ensemble d'emplacement mémoire M', σ : M(P) —> M'. Etant fixé un ordonnancement θ, deux emplacements mémoire m et m' ∈ M(P) θ-interfèrent s'il existe W_1, W_2, R_1, R_2 ∈ M(P) avec $\mu(W_1) = \mu(R_1) = m$, $\mu(W_2) = \mu(R_2) = m'$, θ(W1) << θ(R2), et θ (W2) << θ(R1). On note alors $m ><_\theta m'$. Une allocation mémoire σ est θ-correcte si: $m \triangleright\!\triangleleft_\theta m' \Rightarrow \sigma(m) \neq \sigma(m')$

Un programme transformé est entièrement décrit par la donnée: (i) des opérations Ω(P) du programme P, d'un ordonnancement valide θ, et d'une allocation θ-correcte σ : (Ω(P), θ, σ). Le programme initial peut s'écrire (Ω(P), θ_{seq}, Id), et on a l'équivalence de calcul : (Ω(P), θ_{seq}, Id) = (Ω(P), θ, σ), Id étant l'allocation initiale des données.

Un programme est en assignation unique si chaque opération d'écriture écrit un emplacement mémoire différent, autrement dit si pour W_1, W_2 ε Ω(P), $\sigma(W_1) = \sigma(W_2)$ => $W_1 = W_2$. Pour passer un programme en assignation unique, on se sert de la fonction source s pour lire les bons emplacements On forme M' = {W, W ∈ Ω(P)} et on définit σ par σ(W) = W pour chaque écriture W de Ω(P) et σ(R) = s(R) pour chaque lecture. Le programme transformé (Ω(P), θ, σ) est alors en assignation unique.

A titre d'exemple, pour passer le programme Prog en assignation unique, on commence par calculer la fonction source s(.).

- s((I_1,) : a[0]) = a[0], la source de la lecture a[0] de l'opération (I1,) n'existe pas dans le programme, on garde donc la valeur en entrée de a[0].
- s((I_2,) : a[N - 1]) = a[N -1], pour les mêmes raisons.
- • Pour les lectures de (S, t, i), on obtient les sources suivantes :

$$s((S,t,i):a[i-1]) = \begin{cases} i=1:(I_1) \\ i \geq 2 \wedge t = 0 : a[i-1] \\ i \geq 2 \wedge t \geq 1 : T[t-1][i-1] \end{cases}$$

$$s((S,t,i):a[i]) = \begin{cases} t = 0 : a[i] \\ \\ t \geq 1 : T[t-1][i] \end{cases}$$

$$s((S,t,i):a[i+1]) = \begin{cases} i=N-2:(I_2) \\ i \leq N-2 \wedge t = 0 : a[i+1] \\ i \leq N-2 \wedge t \geq 1 : T[t-1][i+1] \end{cases}$$

Sur les bords (i = 1 ou i = N - 2), la source est donnée par une affectation d'initialisation (I_1 ou I_2). Quand t = 0, la source est la valeur initiale du tableau (comme au premier item). Autrement, la valeur est produite par une instance de T ((T, t - 1, i -1), ou (T, t -1, i) ou (T, t -1, i + 1)).

En applicant l'assignation unique au programme Prog, on obtient alors les lectures du programme Prog en assignation unique, où les lectures sont totalement expansées :

```
I1 : I1 = 0;
I2 : I2 = 0;
     for(t=0; t ≤ K-1; t++)
     {
      for(i=1; i ≤ N-2; i++)
S :     S[t][i] =                                   <--b[i] =a[i-1] + a[i] + a[i+1];
         (i = 1 ? I1 : (t=0 ? a[i-1] : T[t-1][i-1])) +
         (t = 0 ? a[i] : T[t-1][i]) +
         (i = N - 2 ? I2 : (t=0 ? a[i+1] : T[t-1][i+1]));
      for(i=1; i ≤ N-2; i++)
T :     T[t][i] = S[t][i];                          <--a[i] = b[i];
     }
R : R = T[K-1][1];
```

Tel quel (avant avoir appliqué l'allocation (t, i) → (t%1, i%1) à S), le programme ne comporte plus que des dépendances de flot, et peut donc être ordonnancé avec θ(S, t, i) = (1, t + i, t, 0), θ(T), t, i) = (1, t + i, t,1). L'ordonnancement de I_1, I_2 et R reste inchangé.

L'ordre d'exécution est décrit sur la figure 3.

Les intervalles de vie des cases S[t][i] sont disjoints, et donc, on peut appliquer à S l'allocation σ(S[t][i]) = S'[t%1][i%1]. Ci-après est décrit le stade de transformation du programme Prog après l'allocation mémoire sans suivre l'ordre de θ) :

```
I1 : I1 = 0;
I2 : I2 = 0;
    for(t=0; t ≤ K-1; t++)
     {
      for(i=1; i ≤ N-2; i++)
S :  S[t%1][i] =
```

```
        (i = 1 ? I1 : (t=0 ? a[i-1] : T[(t-1)%1][i-1])) +
        (t = 0 ? a[i] : T[t-1][i]) +
        (i = N - 2 ? I2 : (t=0 ? a[i+1] : T[(t-1)%1][i+1]));
    for(i=1; i ≤ N-2; i++)
T :  T[t%1][i] = S[t%1][i];
    }
R : R = T[(K-1)%1][1];
```

Seule une dimension de T est active. La fonction d'allocation (donc l'espace mémoire utilisé) dépend de l'ordonnancement choisi.

Réseaux de processus réguliers

Un réseau de processus réguliers R est la donnée des éléments suivants :

- $\mathcal{P}$, un ensemble de processus.
- Cens = {C_1, ... C_n }, un ensemble de mémoires physiques appelées canaux ; chaque canal C_j, j = 1 à n, est une mémoire tampon adressable. Les canaux sont découpés en sections, nommées cases.
- Un ensemble de connexions de $\mathcal{P}$ x M x $\mathcal{P}$.
- Pour chaque canal C ∈ Cens, l'historique des écritures $\mathcal{W}(C) = W_1^C, ..., W_p^C$ et

  l'historique des lectures $\mathcal{R}(C) = R_1^C , ... , R_p^C$ sont invariants. Cette contrainte implique que la trace d'exécution de chaque processus est fixe. En particulier, les processus peuvent être décrits, mais pas seulement, par des programmes à contrôle statique.
- Un ordre de précédence entre écritures et lectures $\prec_{prec} \subset \mathcal{W}(C) \times \mathcal{R}(R)$. $\prec_{prec}$ est étendu pour exprimer l'ordre séquentiel des écritures ($W_i^C \prec_{seq} W_j^C$ si i < j) et des lectures ($R_i^C \prec_{seq} R_j^C$ si i < j) sur chaque canal C. Dans la suite, on note Ω(R) l'ensemble des opérations exécutées par le réseau R, $\Omega(R) = \bigcup_{C \in Cens} \mathcal{R}(C) \cup \mathcal{W}(C)$.

Un ordonnancement θ_{obs} des opérations de R est observable s'il décrit une exécution possible des opérations de R, autrement dit si pour X, Y ∈ Ω(R), $X \prec_{prec} Y \Rightarrow \theta_{obs}(X) << \theta_{obs}(Y)$.

La compilation d'un programme P à contrôle statique en réseau de processus réguliers est la donnée des éléments suivants :

- Un placement Π : Ω(P) —> $\mathcal{P}$

- Une allocation des canaux σ : Ω(P) —> $\bigcup_{Ci \in C} C_i$
- Un ensemble de synchronisations qui garantissent un ordre de précédence $\prec_{synchro} \subset \Omega(P) x \Omega(P)$ total sur chaque processus Π (X) = Π (Y) ⇒ X $\prec_{synchro}$ Y ou Y $\prec_{prec}$ X et le respect des dépendances de flot W = s(R) ⇒ W $\prec_{préc}$ R .

Chaque affectation du programme P donne lieu à un processus.

Un réseau est partiellement équivalent à P si toutes ses exécutions qui terminent produisent le même calcul que P, c'est à dire si tous les ordonnancements observables dans R totalement définis sur Ω(R) sont des ordonnancements corrects de P. De plus, si le réseau est garanti sans interblocage, on dit qu'il est équivalent à P et que la compilation est correcte.

Réseaux de processus DPN et compilation selon l'invention

Un procédé et un dispositif de synthèse automatique de circuit selon l'invention utilise dans le traitement mis en œuvre par le bloc 2 « Front-End » un réseau de processus particuliers, dit DPN (en anglais, « Data-aware process network »).

Le bloc « Front-End » 2 effectue en effet une compilation d'un programme à contrôle statique qui lui est fourni en entrée, en réseaux de processus DPN.

Un DPN est un réseau de processus régulier qui vérifie les conditions suivantes :

- Chaque canal admet un unique processus source (i.e. un unique processus autorisé à écrire dans le canal) et un unique processus destination (i.e. un unique processus autorisé à lire dans le canal ; dans un mode de réalisation, il existe même un seul port d'entrée d'un unique processus destination).
- Les canaux sont des mémoires bloquantes en écriture et en lecture ; i.e. ce sont des mémoires à accès aléatoire en lecture et écriture telles qu'un processus ne peut y écrire si le contenu alors présent n'a pas été lu et inversement on ne peut les lire tant que le contenu à lire n'a pas été écrit.

Un schéma de compilation (Π, σ, $\prec_{prec}$) d'un programme à contrôle statique P vers un DPN doit vérifier les conditions suivantes:

- Pour chaque lecture dans un canal, la source est bien écrite dans le même canal: si (P_1, C, P_2) est une connexion de P_1 vers P_2 en passant par le canal C, alors Π (R) = P2 ∧ σ(R) ∈ C ⇒ Π (s(R)) = P_1 ∧ σ(s(R)) ∧ C.
- On choisit un ordonnancement correct θ dont l'ordre induit sur Ω(P) est total, et on choisit une allocation σ qui est θ -correcte.
- L'ordre de précédence est total sur chaque processus, et non défini partout

ailleurs,

$$\prec_{prec} = \prec_{\theta} \cap \{(X,Y), \Pi(X) = \Pi(Y)\}.$$

- Pour chaque écriture W, la dernière lecture d(W) = $\max_{\prec_{prec}}$ {R, s(R) = W} libère la case écrite par W, σ(W). La case σ(W) doit alors être à nouveau écrite pour pouvoir être lue.

Une telle compilation est correcte : le DPN obtenu est partiellement équivalent au programme original P et il ne se produit jamais d'interblocage.

Un DPN simple comporte trois types de processus:

- Des processus LD, qui lisent les données entrantes dans la mémoire centrale.
- Des processus ST, qui écrivent le résultat final du calcul en mémoire centrale.
- Des processus Cal en charge du calcul à proprement parler.

On considère un programme P à traduire en DPN par le bloc « Front-End » 2 et un ordonnancement θ valide.

Afin de construire facilement les processus LD et ST, le bloc « Front-End » 2 ajoute au programme P, pour chaque tableau de données *a* lu en entrée dans la mémoire principale distante, la famille d'opérations suivante :

$\vec{i} \in \Omega$: a[$\vec{i}$] = 0 ;//(LD_a, $\vec{i}$), où Ω est l'ensemble des indices du tableau a.

Pour chaque tableau écrit *b*, le bloc « Front-End » 2 ajoute au programme la famille d'opérations suivante, où *d* est une variable scalaire quelconque:

$\vec{i} \in \Omega$: d = b[$\vec{i}$] ; //(ST_b, $\vec{i}$) où Ω est l'ensemble des indices du tableau b.

On convient que les opérations LD_a sont exécutées tout au début de P, et que les opérations de ST_b sont exécutées tout à la fin de P. Ainsi (LD_a, $\vec{i_1}$) $\prec_{prec}$ (S, $\vec{i_2}$) $\prec_{prec}$ (ST_b, $\vec{i_3}$) pour chaque opération originale (S, $\vec{i_2}$) de P. Cette propriété peut être spécifiée avec un ordonnancement multidimensionnel.

Le bloc « Front-End » 2 crée un réseau DPN en plaçant chaque affectation sur un processus différent, P = {S1, ..., S_n} (affectations de P), et Π(W) = Π (R1) = ... = Π (R_n) = S, où W, R1, …, Rn sont les opérations réalisées par l'affectation S : W = f(R1, …, Rn) . En particulier, il est créé un processus L_{Da} par tableau *a* lu, et un processus S_{Tb} par tableau *b* écrit. La connexion de ces processus aux processus de calcul s'obtient naturellement avec la fonction source *s*.

Chaque processus est muni par le bloc « Front-End » 2 des éléments suivants:

- *Un port d'entrée* par référence lue par l'affectation implémentée par le processus. Le port d'entrée est un *multiplexeur,* chargé de récupérer la valeur de la référence dans le bon canal.
- *Un port de sortie*, qui émet la valeur calculée par le processus. Le port de sortie est un *démultiplexeur* qui écrit la valeur calculée par le processus dans le bon canal.

Pour construire un DPN, il suffit au bloc « Front-End » 2 de calculer ces éléments, et d'allouer chaque canal (fonction σ).

Multiplexage, démultiplexage

Lors de la compilation d'un programme P, le bloc « Front-End » 2 calcule la fonction source pour chaque lecture.

Pour une opération $(T, \vec{i}) : \vec{i} \in D$ (domaine d'itération de T) : $W = f(R1, ..., R_n)$, la source d'une lecture R parmi {R1, …, Rn} se présente sous la forme :

$$s((T, \vec{i}) : R) = \begin{cases} \vec{i} \in D_1 : (S_1, u_1(\vec{i})) \\ \dots \\ \vec{i} \in D_\ell : (S_\ell, u_\ell(\vec{i})) \end{cases}$$

où les D_k sont des polyèdres convexes fermés, les u_k sont des fonctions affines et les T_k sont des affectations du programme.

Pour chaque couple d'affectations (S, T) telles que *S* écrit un tableau lu par la référence R de T, le bloc « Front-End » 2 crée un canal $C_{S,T,R}$. Si *D* est le domaine d'itération de S, $C_{S,T,R}$ est un tableau de dimension égale à la dimension de D, dim D.

Ces canaux sont ensuite connectés par le bloc « Front-End » 2 aux processus *via* les ports d'entrée et de sortie en procédant de la façon suivante.

Pour chaque affectation $T \in \{T_1, ..., T_n\}$, pour chaque lecture R de T, pour chaque clause $\vec{i} \in D_k : (S_k, u_k(i))$ de la source s((T,i) : R), le bloc « Front-End » 2 effectue les opérations suivantes :

- ajout au port d'entrée correspondant à R d'une connexion au canal $C_{Sk,T,R}$, et on ajoute la clause de multiplexage $\vec{i} \in D_k : C_{SkT,R}[u_k(\vec{i})]$ qui commande la lecture de la case $u_k(\vec{i})$ du canal $C_{Sk,T,R}$ quand l'itération courante vérifie $\vec{i} \in D_k$;
- ajout au port de sortie de S_k une connexion vers le canal $C_{Sk,T,R}$, et ajout de la

clause de multiplexage $\vec{i} \in u_k(D_k)$: $C_{Sk,T,R}[\vec{i}]$ qui commande l'écriture de la valeur calculée dans la case $\vec{i}$ (vecteur d'itération courant) du canal $C_{Sk,T,R.}$ quand l'itération courante vérifie $\vec{i} \in u_k(D_k)$.

Par construction, chaque canal $C_{Sk,T,R}$ possède bien une unique source (S_k) et un seul port d'une unique destination (*T*).

Un multiplexeur pour chaque port d'entrée d'une affectation T est adapté pour déterminer la clause de multiplexage correspondant au vecteur d'itération courant de l'affectation et fournissant le canal pertinent et sa case pour ce vecteur d'itération et pour lire cette case de ce canal.

Un démultiplexeur pour chaque port de sortie d'une affectation S est adapté pour déterminer la clause de multiplexage correspondant au vecteur d'itération courant de l'affectation et fournissant le canal pertinent et sa case pour ce vecteur d'itération, et pour écrire le résultat dans cette case de ce canal.

Ensuite, les processus LD et ST sont connectés au DPN. En effet, lorsque T_k est un processus LD_a, il s'agit d'une valeur d'entrée du processus, et la compilation par le bloc Front-End 2 crée naturellement le multiplexage pour récupérer la bonne valeur dans le canal écrit par LD_a.

De plus, le multiplexeur d'un processus de sortie ST_b récupère dans ses canaux les valeurs finales pour chaque case de *b.*

A titre d'illustration, la figure 4 illustre le résultat de la compilation par le bloc « Front-End » 2 du programme Prog. Les processus sont représentés par des boîtes aux contours en trait gras. Les processus de calcul sont représentés par des boîtes aux contours en trait gras sans motif intérieur, les processus LD_a et ST (LOAD et STORE) sont représentés par des boîtes aux contours en trait gras avec un motif intérieur strié. Les canaux sont représentés par des boîtes aux contours en trait fin avec des points en motif intérieur, les multiplexeurs et démultiplexeurs sont représentés en contours en trait fin sans motif intérieur.

Chaque processus de calcul possède un ou plusieurs ports d'entrée. Chaque port d'entrée est disposé entre un multiplexeur qui lui est propre, nommé sur la figure 4 selon les processus « MUX », « MUX1 », « MUX2 », « MUX3 », et le processus.

Le multiplexeur détermine quel est le canal pertinent et la case du canal pour l'itération courante, et lit la donnée dans cette case du canal pertinent.

De même, chaque processus possède un port de sortie, qui transmet le résultat du

calcul courant vers un démultiplexeur, nommé sur la figure 4 « DEMUX ». Le démultiplexeur détermine quel est le canal pertinent ou les canaux pertinents, et la case du canal pour l'itération courante de l'affectation et écrit le résultat dans un ou plusieurs canaux.

Par exemple, le processus, dans le programme Prog, calculant l'affectation S correspondant à b[i] = a[i-1] + a[i] + a[i+1], possède trois ports d'entrée, avec les valeurs respectives de a[i - 1], a[i], et a[i + 1]. Pour la lecture de a[i-1], le multiplexeur MUX 1 détermine à l'aide de la fonction source s((S, t, i) : a[i-1]) que quand $i \geq 2 \wedge t = 0$, la source est (LD_a, i - 1), d'où la sélection d'un canal en provenance de LOAD(a). Le démultiplexage correspondant du processus LOAD(a) est donné par $[(t, i) \rightarrow i - 1] (i \geq 2 \Lambda t = 0) = i \geq 1$.

Le DPN a été obtenu et opère de façon analogue pour chaque source de chaque lecture de chaque affectation.

Dans la suite, le processus qui exécute l'affectation *S* sera dénommé « processus S » et le processus qui exécute l'affectation *T* sera dénommé « processus *T* ».

Allocation des canaux

Le bloc « Front-End » 2 est en outre adapté pour créer, lors de la compilation du réseau DPN pour un programme P, une fonction d'allocation σ pour chaque canal.

σ est utilisée comme fonction d'adressage du canal. Toute requête vers Canal [$\vec{i}$] accède en réalité à Canal [σ($\vec{i}$)] .

L'allocation se présente sous la forme d'une fonction affine modulaire $A\vec{i} \mod \vec{b}$. Les coordonnées de $\vec{b}$ correspondent aux tailles des différentes dimensions du canal $C_{Tk,S,Rk}$, qui peut maintenant être effectivement construit par le bloc Front-end 2.

A titre d'illustration relative à la construction d'un réseau DPN pour le programme Prog, pour allouer le canal C entre *S* et *T, avec*

θ(S, t, i) = (t + i, t, 0), θ(T, t, i) = (t + i, t, 1)**.** le Bloc front-end 2 calcule la fonction d'allocation σ(t, i) = (t mod 1, i mod 1), par exemple comme décrit dans « Lattice-Based Array Contraction: From Theory to Practice », Christophe Alias, Fabrice Baray, Alain Darte, Research Report RR2007-44, Laboratoire de l'Informatique du Parallélisme, ENS-Lyon, November 2007. En d'autres termes, $C_{S,T,b[i]}$ est un canal à 2 dimensions de taille 1 x 1, qui est dorénavant accédé avec σ.

Il peut être implémenté avec un simple registre.

Synchronisation des canaux

Dans un réseau DPN selon l'invention, les canaux sont bloquants en lecture et en écriture : une case d'un canal ne peut pas être lue avant d'avoir été écrite (par sa source), et une case ne pas être écrite avant que sa valeur couramment stockée ne soit lue pour la dernière fois.

Les écritures et les lectures dans un canal sont synchronisées, selon l'invention, pour forcer cette propriété.

Le bloc « Front-End » 2 est ainsi adapté pour construire une *unité de synchronisation* S_c, pour chaque canal *c* qui relie le port de sortie d'un processus producteur *P* au port d'entrée R_k d'un processus consommateur *C*.

Chaque itération *W* (écriture) du producteur et chaque itération *R* (lecture) du consommateur dans le canal c donnent lieu, préalablement à leur exécution, à la fourniture d'une requête d'exécution soumises à l'unité de synchronisation S_c, qui décide d'autoriser ou non leur exécution.

Pour forcer les dépendances de flot, l'unité de synchronisation décide dans un mode de réalisation de l'invention, d'arrêter l'exécution du consommateur si la donnée qu'il veut lire n'est pas encore produite. Et pour forcer les dépendances anti-, l'unité de synchronisation décide, dans un mode de réalisation de l'invention, d'arrêter l'exécution du producteur s'il veut écraser une donnée qui n'a pas encore été lue pour la dernière fois. Notons qu'il n'est pas nécessaire de forcer les dépendances de sortie, toutes les écritures étant exécutées par le producteur dans l'ordre séquentiel de l'ordonnancement.

Ces deux évènements se traduisent de la façon suivante par exemple par l'utilisation par l'unité de synchronisation de la fonction suivante S_c par l'unité de synchronisation, où X $\preceq_\theta$ Y signifie que $\theta(X) \lll \theta(Y)$ (i.e. $\theta(X)$ est antérieur ou égal à $\theta(Y)$) et s() est la fonction source, pour déterminer si la réponse à une requête d'un processus consommateur C est un signal Freeze(C) destiné au processus C ou pour déterminer si la réponse à une requête d'un processus producteur P est un signal Freeze(P) destiné au processus P :

$$S_c(W,R) = \begin{cases} Freeze(C) & si \quad W \preceq_\theta s(R) \\ Freeze(P) & si \quad \{R' \in \Omega(P,I), R \prec_\theta R' \prec_\theta W \wedge \sigma_c(R') = \sigma_c(W)\} \neq \emptyset \end{cases}$$

Le signal *Freeze* (*C*) interrompt l'exécution du processus consommateur C, et Freeze (*P*) celle du producteur P.

Dans un mode de réalisation, un signal *Freeze* existe aussi longtemps que sa condition est vraie. Lorsque sa condition redevient fausse, il cesse d'être émis, et le

processus reprend son exécution.

Dans un autre mode de réalisation, un signal Freeze est émis, une fois, en réponse à une requête d'un processus, lors que la condition testée est vraie. Et lorsque la condition est fausse ou dès qu'elle devient fausse, un signal d'autorisation est émis vers le processus.

La première condition dans la formule de S_c ci-dessus signifie que l'exécution du consommateur est interrompue lorsque la donnée à lire n'est pas encore écrite. Autrement dit, lorsque le producteur P exécute une écriture *W* qui précède l'écriture de la donnée lue par *R, s(R)*.

La deuxième condition signifie que l'exécution du producteur P est interrompue quand il est sur le point d'écraser une donnée qui n'a pas fini d'être lue.

Autrement dit, lorsque le producteur P tente d'écrire une case $\sigma_c(W)$ qui doit encore être lue plus tard par *R'*.

La fonction source *s(R)* est déjà calculée par le multiplexeur du processus *C* qui lit le canal *c*.

La condition *Freeze (P)* peut être simplifiée de manière conservative en $R \prec_\theta W$ (en effet, si *Freeze (P)* est vrai, on a nécessairement $R \prec_\theta W$).

Preuve. Si Freeze (*P*) est vrai, il existe R'' avec $R \prec_\theta R' \prec_\theta W$,. On obtient le résultat par transitivité de $\prec_\theta$,

Ce théorème signifie que chaque fois qu'il faut bloquer le producteur, $R \prec_\theta W$, devient vrai. Par contre, $R \prec_\theta W$, peut continuer à être vrai plusieurs itérations après que la case à écrire ait été libérée par le consommateur bloquant inutilement le producteur. En particulier, quand $d(R') \prec_\theta R \prec_\theta W$, le producteur devra encore attendre que R soit exécuté. Au final, le producteur respectera exactement l'ordonnancement θ spécifié par l'utilisateur, sans "faire de zèle ». Il est donc de la responsabilité de l'utilisateur de fournir l'ordonnancement le plus parallèle possible.

L'unité de synchronisation S_c respecte les dépendances de données et ne produit jamais d'interblocage.

A titre d'illustration, en référence à la figure 5, considérons à nouveau le canal $c = C_{S,T,b[i]}$ entre les processus S (*producteur)* et T (*consommateur*).

Le processus consommateur T, avant d'entamer chaque nouvelle itération (t_C, i_C), transmet à l'unité de synchronisation du canal c dans lequel son multiplexeur a déterminé

qu'il doit lire une référence, une requête d'autorisation d'exécution d'une nouvelle itération, l'identifiant de sa nouvelle itération, (t_C, i_C), ainsi que l'identifiant de l'itération correspondante attendue (t_S, i_S) du processeur producteur S, i.e. qui donne lieu à l'écriture dans le canal c de cette référence. (t_S, i_S) est obtenue par la fonction source s, d'où l'indice 's'. Le processus consommateur n'entame pas cette nouvelle itération tant qu'il n'y a pas été autorisé par l'unité de synchronisation.

Similairement, le processus producteur S, avant d'entamer chaque nouvelle itération (t_P, i_P), transmet à l'unité de synchronisation du canal c dans lequel son démultiplexeur a déterminé qu'il doit écrire un résultat, une requête d'autorisation d'exécution d'une nouvelle itération et l'identifiant de sa nouvelle itération, (t_P, i_P). Le processus producteur n'entame pas cette nouvelle itération tant qu'il n'y a pas été autorisé par l'unité de synchronisation.

Considérons que l'unité de synchronisation vient de recevoir la requête d'autorisation d'exécution d'une nouvelle itération du processus consommateur T, (t_C, i_C), l'identifiant de sa nouvelle itération, (t_C, i_C), et l'identifiant de l'itération correspondante attendue (t_S, i_S) du processeur producteur S.

Si $\theta(S,t_S,i_S) << \theta(S,t_P,i_P)$,((t_P, i_P) étant le dernier identifiant d'itération du processus producteur S alors transmis par S à l'unité de synchronisation) et θ la fonction d'ordonnancement, l'unité de synchronisation autorise la nouvelle itération (t_C, i_C) du consommateur T. Une commande correspondante Smd_T est alors transmise en réponse par l'unité de synchronisation au processus consommateur T (comme vu précédemment, cette commande est alors un signal d'autorisation ou l'absence d'un signal de type Freeze(C) suivant les modes de réalisation évoqués).

Si au contraire $\theta(S,t_P,i_P) \underline{<<} \theta(S,t_S,i_S)$, et tant que cette dernière expression reste vraie ((t_P, i_P) étant le dernier identifiant d'itération du processus producteur S alors transmis par S à l'unité de synchronisation), l'unité de synchronisation n'autorise pas la nouvelle itération (t_C, i_C) du consommateur T. Une commande correspondante Smd_T est alors transmise en réponse par l'unité de synchronisation au processus consommateur T (comme vu précédemment, cette commande est alors un signal Freeze(C), continu ou ponctuel suivant les modes de réalisation évoqués).

Considérons à présent que l'unité de synchronisation vient de recevoir la requête d'autorisation d'exécution d'une nouvelle itération du processus producteur S et l'identifiant de sa nouvelle itération, (t_P, i_P).

Si $\theta(S,t_P,i_P) \underline{\underline{<<}} \theta(T,t_C,i_C)$,((tC, iC) étant le dernier identifiant d'itération du processus consommateur T alors transmis par T à l'unité de synchronisation), l'unité de synchronisation autorise la nouvelle itération (t_P, i_P) du producteur S. Une commande correspondante Smd_S est alors transmise en réponse par l'unité de synchronisation au processus producteur S (comme vu précédemment, cette commande est alors un signal d'autorisation ou l'absence d'un signal de type Freeze(P) suivant les modes de réalisation évoqués).

Si au contraire $\theta(T,t_C,i_C) << \theta(S,t_P,i_P)$, et tant que cette dernière expression reste vraie, l'unité de synchronisation n'autorise pas la nouvelle itération (t_P, i_P) du processus producteur S. Une commande correspondante Smd_S est alors transmise en réponse par l'unité de synchronisation au processus consommateur T (comme vu précédemment, cette commande est alors un signal Freeze(P), continu ou ponctuel suivant les modes de réalisation évoqués).

En prenant comme fonction d'ordonnancement celle indiquée dans les exemples précédents relatif à la construction d'un réseau DPN pour le programme Prog, S étant le processus producteur et T étant le processus consommateur, les expressions relatives à Freeze(C) et Freeze(P) indiquée plus haut deviennent :

Freeze(C) si, et tant que, $(t_P + i_P, t_P, 0) \underline{\underline{<<}} (t_S + i_S, t_S, 0)$

i.e. si et tant que : $(t_P + i_P < t_s + i_S) \vee (t_P + i_P = t_S + i_S \wedge t_P < t_S) \vee (t_P + i_P = t_S + i_S \wedge t_P = t_S)$;

$\vee$ signifiant « OU ».

Freeze(P) si, et tant que, $(t_C + i_C, t_C, 1) << (t_P + i_P, t_P, 0)$

i.e. si et tant que : $(t_C + i_C < t_P + i_P) \vee (t_C + i_C = t_P + i_P \wedge t_C < t_P)$.

Par nature, l'ordre lexicographique (<<) produit des expressions très redondantes, ce qui permet de produire un circuit de test très simple.

On notera que dans d'autres modes de réalisation, des règles de synchronisation autres sont mises en œuvre par l'unité de synchronisation pour synchroniser entre eux les processus producteur et consommateur associés à un même canal.

Une fois la transformation en représentation intermédiaire DPN ainsi réalisée par le bloc 2 « Front-End » à partir de la représentation algorithmique du programme Prog, cette représentation intermédiaire est fournie au bloc 3 « Back-End ».

De manière connue, le bloc 3 « Back-End » construit matériellement le bloc de

circuits électroniques à partir de cette représentation intermédiaire DPN On pourra utiliser, par exemple, un outil de synthèse existant comme C2H, comme décrit dans « Christophe Alias, Alain Darte, and Alexandru Plesco. Optimizing DDR-SDRAM Communications at C-Level for Automatically-Generated Hardware Accelerators. An Experience with the Altera C2H HLS Tool. In 21st IEEE International Conference on Application-specific Systems, Architectures and Processors (ASAP'10), Rennes, France, pages 329-332, July 2010. IEEE Computer Society ». Chaque processus, canal, multiplexeur est décrit comme une fonction C2H. Chaque fonction est traduite par C2H dans un module matériel. Les connexions entre les modules sont spécifiées à C2H en utilisant des pragmas de connexion. Chaque connexion sera traduit par C2H dans un bus matériel standard.

Un dispositif de synthèse automatique de circuits selon l'invention permet de générer automatiquement des circuits électroniques réalisant, rapidement et en requérant un volume physique occupé limité, les calculs d'un programme P écrit dans un langage de haut niveau.

Il existe de nombreux modèles d'exécution parallèle sous forme de réseaux de processus (KPN, SDF, etc.). L'invention propose la compilation en réseaux de processus DPN (Data-aware Process Network) selon un modèle d'exécution parallèle adapté aux contraintes matérielles de la synthèse de circuits dans lequel les transferts de données avec une mémoire distante sont explicites. Une telle compilation en DPN est consistante en cela qu'elle transforme un programme séquentiel en un réseau de processus DPN équivalent, garanti sans interblocage. Les entrées/sorties (i.e. les transferts de données avec une mémoire distante) sont explicites et les canaux de communication sont des mémoires tampons beaucoup moins contraignantes que des FIFO, pilotées des unités de synchronisation de performance équivalente.

REVENDICATIONS

1.- Procédé de synthèse automatique de circuits comprenant une étape de compilation d'un programme logiciel en un réseau de processus réguliers interconnectés à l'aide de canaux (c), selon lequel chaque canal est une mémoire physique adressable, chacun desdits processus applique, lors de chaque itération d'un domaine d'itérations qui lui est associé, au moins une lecture d'une donnée dans un canal et/ou une écriture d'une donnée dans un canal, et une fonction d'ordonnancement fournit, pour chaque valeur d'itération de chaque processus, une position d'exécution du processus sélectionnée dans un ensemble temporellement ordonné de positions d'exécution ;

ledit procédé de synthèse de circuits étant caractérisé en ce qu'il comprend les étapes suivantes, lors de l'étape de compilation, pour chacun desdits canaux (c) :

un unique processus, dit processus producteur (S), est autorisé à écrire des données dans le canal (c) et un unique processus, dit processus consommateur (T), est autorisé à lire des données dans ledit canal ; et

une unité de synchronisation (S_c) est associée audit canal, et telle qu'elle collecte, préalablement au lancement de chaque nouvelle itération par le processus producteur du canal, la valeur (t_p, i_p) de ladite nouvelle itération du processus producteur, et collecte, préalablement au lancement de chaque nouvelle itération par le processus consommateur du canal, la valeur d'itération du processus consommateur (t_C, i_C); ladite unité de synchronisation, suite à la collecte de la valeur d'une nouvelle itération du processus producteur, respectivement consommateur, autorisant ou gelant la mise en œuvre de ladite nouvelle itération dudit processus producteur, respectivement consommateur, en fonction d'une comparaison d'une position d'exécution déterminée en fonction de la valeur de nouvelle itération collectée du processus producteur, respectivement consommateur, et d'une position d'exécution déterminée en fonction de la dernière valeur d'itération collectée du processus consommateur, respectivement producteur.

2.- Procédé de synthèse automatique de circuits selon la revendication 1, selon lequel ladite unité de synchronisation (S_c) est telle qu'elle pilote le fonctionnement du processus producteur (S) dudit canal (c) en mettant en œuvre les étapes suivantes suite à la collecte de la valeur d'une nouvelle itération (t_p, i_p) dudit processus producteur :

- l'unité de synchronisation compare la nouvelle position d'exécution ($\theta(t_P, i_P)$) de ladite valeur collectée, à la position d'exécution ($\theta(t_C, i_C)$) de la dernière valeur d'itération du processus consommateur (T) collectée ; et

- ladite unité de synchronisation gèle la mise en œuvre par le processus producteur de cette nouvelle itération si la position d'exécution ($\theta(t_P, i_P)$) de cette nouvelle itération est postérieure à la position d'exécution ($\theta(t_C, i_C)$) de la dernière valeur d'itération collectée du processus consommateur, le gel étant maintenu par l'unité de synchronisation jusqu'à la collecte d'une future valeur d'itération du processus consommateur telle que la position d'exécution ($\theta(t_C, i_C)$) de cette future valeur d'itération soit postérieure à ladite position d'exécution ($\theta(t_P, i_P)$) de la nouvelle itération du processus consommateur.

3.- Procédé de synthèse automatique de circuits selon la revendication 1 ou 2, selon lequel ladite unité de synchronisation (S_c) est telle qu'elle pilote le fonctionnement du processus consommateur (T) dudit canal en mettant en œuvre les étapes suivantes suite à la collecte de la valeur d'une nouvelle itération dudit processus consommateur :

- l'unité de synchronisation détermine la position d'exécution ($\theta(t_S, i_S)$) de celle des valeurs d'itération du processus producteur (S) qui donne lieu à l'écriture dans ledit canal (c) de la donnée qui doit être lue par le processus consommateur lors de cette nouvelle itération ; et
- ladite unité de synchronisation gèle la mise en œuvre par le processus consommateur de sa nouvelle itération si la position d'exécution ($\theta(t_S, i_S)$) déterminée est postérieure ou égale à la position d'exécution ($\theta(t_P, i_P)$) de la dernière valeur d'itération collectée du processus producteur, le gel étant maintenu par l'unité de synchronisation jusqu'à la collecte d'une future valeur d'itération du processus producteur telle que la position d'exécution ($\theta(t_P, i_P)$) de cette future valeur d'itération soit supérieure strictement à ladite position d'exécution ($\theta(t_S, i_S)$) déterminée.

4.- Procédé de synthèse automatique de circuits selon l'une des revendications 1 à 3, selon lequel le processus consommateur (T) ou producteur (S) de chaque canal après la fin d'une précédente itération dudit processus et préalablement au lancement de la nouvelle itération succédant à la précédente itération, délivre à l'unité de synchronisation (S_c) du canal (c), la valeur d'itération de la nouvelle itération du processus, et met en œuvre cette nouvelle itération du processus en fonction d'une commande de pilotage de l'unité de supervision indiquant un gel ou une autorisation.

5.- Procédé de synthèse de circuits selon la revendication 4, selon lequel le processus consommateur (T) du canal (c), après la fin d'une précédente itération dudit processus et préalablement au lancement de la nouvelle itération succédant à la précédente itération, délivre en outre à l'unité de synchronisation (S_c) du canal, ladite valeur d'itération du processus producteur (S) qui donne lieu à l'écriture dans ledit canal de la donnée qui doit être lue par le processus consommateur lors de cette nouvelle itération.

6.- Dispositif (1) de synthèse automatique de circuits adapté pour effectuer une compilation d'un programme logiciel en générant un réseau de processus réguliers interconnectés à l'aide de canaux (c), dans lequel chaque canal est une mémoire physique adressable, chacun desdits processus étant adapté pour appliquer, lors de chaque itération d'un domaine d'itérations qui lui est associé, au moins une lecture d'une donnée dans un canal et/ou une écriture d'une donnée dans un canal, et une fonction d'ordonnancement étant adaptée pour fournir, pour chaque valeur d'itération de chaque processus, une position d'exécution du processus sélectionnée dans un ensemble temporellement ordonné de positions d'exécution ;

ledit dispositif de synthèse de circuits étant caractérisé en ce qu'il est adapté pour lors de ladite compilation, générer et associer à chacun desdits canaux (c) :

un unique processus, dit processus producteur (S), autorisé à écrire des données dans le canal (c) et un unique processus, dit processus consommateur (T), autorisé à lire des données dans ledit canal ; et

une unité de synchronisation (S_c) associée audit canal et adaptée pour collecter, préalablement au lancement de chaque nouvelle itération par le processus producteur du canal, la valeur (t_p, i_p) de ladite nouvelle itération du processus producteur, et collecter, préalablement au lancement de chaque nouvelle itération par le processus consommateur du canal, la valeur d'itération du processus consommateur (t_C, i_C) ; ladite unité de synchronisation, suite à la collecte de la valeur d'une nouvelle itération du processus producteur, respectivement consommateur, étant adaptée pour autoriser ou geler la mise en œuvre de ladite nouvelle itération dudit processus producteur, respectivement consommateur, en fonction d'une comparaison d'une position d'exécution déterminée en fonction de la valeur de nouvelle itération collectée du processus producteur, respectivement consommateur, et d'une position d'exécution déterminée en fonction de la dernière valeur d'itération collectée du processus consommateur, respectivement producteur.

7.- Dispositif (1) de synthèse automatique de circuits selon la revendication 6, dans lequel ladite unité de synchronisation (S_c) générée est adaptée pour piloter le fonctionnement du processus producteur (S) dudit canal (c) en mettant en œuvre les étapes suivantes suite à la collecte de la valeur d'une nouvelle itération (t_p, i_p) dudit processus producteur :

- comparer la nouvelle position d'exécution ($\theta(t_P, i_P)$) de ladite valeur collectée, à la position d'exécution ($\theta(t_C, i_C)$) de la dernière valeur d'itération du processus consommateur (T) collectée ; et
- geler la mise en œuvre par le processus producteur de cette nouvelle itération si la position d'exécution ($\theta(t_P, i_P)$) de cette nouvelle itération est postérieure à la position d'exécution ($\theta(t_C, i_C)$) de la dernière valeur d'itération collectée du processus consommateur, et maintenir le gel jusqu'à la collecte d'une future valeur d'itération du processus consommateur telle que la position d'exécution ($\theta(t_C, i_C)$) de cette future valeur d'itération soit postérieure à ladite position d'exécution ($\theta(t_P, i_P)$) de la nouvelle itération du processus consommateur.

8.- Dispositif (1) de synthèse automatique de circuits selon la revendication 6 ou 7, ladite unité de synchronisation (S_c) générée est adaptée pour piloter le fonctionnement du processus consommateur (T) dudit canal en mettant en œuvre les étapes suivantes suite à la collecte de la valeur d'une nouvelle itération dudit processus consommateur :

- déterminer la position d'exécution ($\theta(t_S, i_S)$) de celle des valeurs d'itération du processus producteur (S) qui donne lieu à l'écriture dans ledit canal (c) de la donnée qui doit être lue par le processus consommateur lors de cette nouvelle itération, ; et
- geler la mise en œuvre par le processus consommateur de sa nouvelle itération si la position d'exécution ($\theta(t_S, i_S)$) déterminée est postérieure ou égale à la position d'exécution ($\theta(t_P, i_P)$) de la dernière valeur d'itération collectée du processus producteur, et maintenir le gel jusqu'à la collecte d'une future valeur d'itération du processus producteur telle que la position d'exécution ($\theta(t_P, i_P)$) de cette future valeur d'itération soit supérieure strictement à ladite position d'exécution ($\theta(t_S, i_S)$) déterminée.

9.- Dispositif de synthèse automatique de circuits selon l'une des revendications 6 à 8, dans lequel le processus généré consommateur (T) ou producteur (S) de chaque

canal, est adapté pour, après la fin d'une précédente itération dudit processus et préalablement au lancement de la nouvelle itération succédant à la précédente itération, délivrer à l'unité de synchronisation (S_c) du canal (c), la valeur d'itération de la nouvelle itération du processus, et mettre en œuvre cette nouvelle itération du processus en fonction d'une commande de pilotage de l'unité de supervision indiquant un gel ou une autorisation.

10.- Dispositif de synthèse de circuits selon la revendication 9, dans lequel le processus consommateur (T) du canal (c), après la fin d'une précédente itération dudit processus et préalablement au lancement de la nouvelle itération succédant à la précédente itération, est adapté pour délivrer en outre à l'unité de synchronisation (S_c) du canal, ladite valeur d'itération du processus producteur (S) qui donne lieu à l'écriture dans ledit canal de la donnée qui doit être lue par le processus consommateur lors de cette nouvelle itération.

11.- Circuit synthétisé par exemple suite à la mise en œuvre d'un procédé de synthèse automatique de circuit selon l'une des revendications 1 à 5, comprenant un réseau de processus réguliers interconnectés à l'aide de canaux (c), dans lequel chaque canal est une mémoire physique adressable, chacun desdits processus étant adapté pour appliquer, lors de chaque itération d'un domaine d'itérations qui lui est associé, au moins une lecture d'une donnée dans un canal et/ou une écriture d'une donnée dans un canal, et une fonction d'ordonnancement étant adaptée pour fournir, pour chaque valeur d'itération de chaque processus, une position d'exécution du processus sélectionnée dans un ensemble temporellement ordonné de positions d'exécution ;

ledit circuit étant caractérisé en ce qu'il comprend :

un unique processus, dit processus producteur (S), associé à chaque canal et autorisé à écrire des données dans le canal (c) et un unique processus, dit processus consommateur (T), associé à chaque canal et autorisé à lire des données dans ledit canal ; et

une unité de synchronisation (S_c) associée audit canal et adaptée pour collecter, préalablement au lancement de chaque nouvelle itération par le processus producteur du canal, la valeur (t_p, i_p) de ladite nouvelle itération du processus producteur, et collecter, préalablement au lancement de chaque nouvelle itération par le processus consommateur du canal, la valeur d'itération du processus consommateur (t_C, i_C) ; ladite unité de synchronisation, suite à la collecte de la valeur d'une nouvelle itération du processus

producteur, respectivement consommateur, étant adaptée pour autoriser ou geler la mise en œuvre de ladite nouvelle itération dudit processus producteur, respectivement consommateur, en fonction d'une comparaison d'une position d'exécution déterminée en fonction de la valeur de nouvelle itération collectée du processus producteur, respectivement consommateur, et d'une position d'exécution déterminée en fonction de la dernière valeur d'itération collectée du processus consommateur, respectivement producteur.

12.- Circuit synthétisé selon la revendication 11, dans lequel ladite unité de synchronisation (S_c) générée est adaptée pour piloter le fonctionnement du processus producteur (S) dudit canal (c) en mettant en œuvre les étapes suivantes suite à la collecte de la valeur d'une nouvelle itération (t_p, i_p) dudit processus producteur :

- comparer la nouvelle position d'exécution ($\theta(t_P, i_P)$) de ladite valeur collectée, à la position d'exécution ($\theta(t_C, i_C)$) de la dernière valeur d'itération du processus consommateur (T) collectée ; et
- geler la mise en œuvre par le processus producteur de cette nouvelle itération si la position d'exécution ($\theta(t_P, i_P)$) de cette nouvelle itération est postérieure à la position d'exécution ($\theta(t_C, i_C)$) de la dernière valeur d'itération collectée du processus consommateur, et maintenir le gel jusqu'à la collecte d'une future valeur d'itération du processus consommateur telle que la position d'exécution ($\theta(t_C, i_C)$) de cette future valeur d'itération soit postérieure à ladite position d'exécution ($\theta(t_P, i_P)$) de la nouvelle itération du processus consommateur.

13.- Circuit synthétisé selon la revendication 11 ou 12, ladite unité de synchronisation (S_c) générée est adaptée pour piloter le fonctionnement du processus consommateur (T) dudit canal en mettant en œuvre les étapes suivantes suite à la collecte de la valeur d'une nouvelle itération dudit processus consommateur :

- déterminer la position d'exécution ($\theta(t_S, i_S)$) de celle des valeurs d'itération du processus producteur (S) qui donne lieu à l'écriture dans ledit canal (c) de la donnée qui doit être lue par le processus consommateur lors de cette nouvelle itération, ; et
- geler la mise en œuvre par le processus consommateur de sa nouvelle itération si la position d'exécution ($\theta(t_S, i_S)$) déterminée est postérieure ou égale à la position d'exécution ($\theta(t_P, i_P)$) de la dernière valeur d'itération

collectée du processus producteur, et maintenir le gel jusqu'à la collecte d'une future valeur d'itération du processus producteur telle que la position d'exécution ($\theta(t_P, i_P)$) de cette future valeur d'itération soit supérieure strictement à ladite position d'exécution ($\theta(t_S, i_S)$) déterminée.

14.- Circuit synthétisé selon l'une des revendications 11 à 13, dans lequel le processus généré consommateur (T) ou producteur (S) de chaque canal, est adapté pour, après la fin d'une précédente itération dudit processus et préalablement au lancement de la nouvelle itération succédant à la précédente itération, délivrer à l'unité de synchronisation (S_c) du canal (c), la valeur d'itération de la nouvelle itération du processus, et mettre en œuvre cette nouvelle itération du processus en fonction d'une commande de pilotage de l'unité de supervision indiquant un gel ou une autorisation.

15.- Circuit synthétisé selon la revendication 14, dans lequel le processus consommateur (T) du canal (c), après la fin d'une précédente itération dudit processus et préalablement au lancement de la nouvelle itération succédant à la précédente itération, est adapté pour délivrer en outre à l'unité de synchronisation (Sc) du canal, ladite valeur d'itération du processus producteur (S) qui donne lieu à l'écriture dans ledit canal de la donnée qui doit être lue par le processus consommateur lors de cette nouvelle itération.

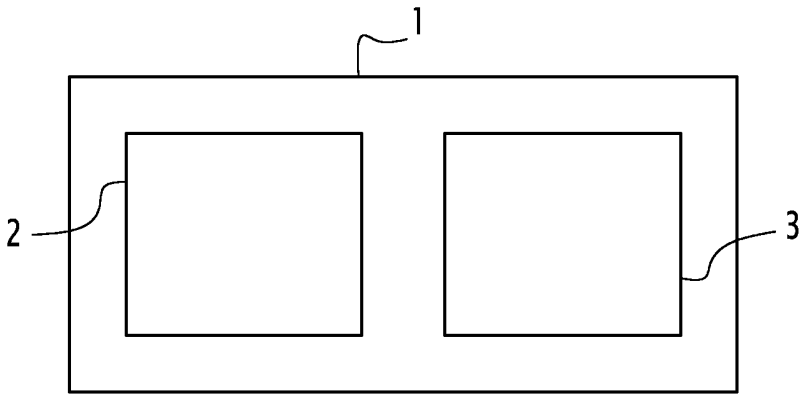


FIG.1

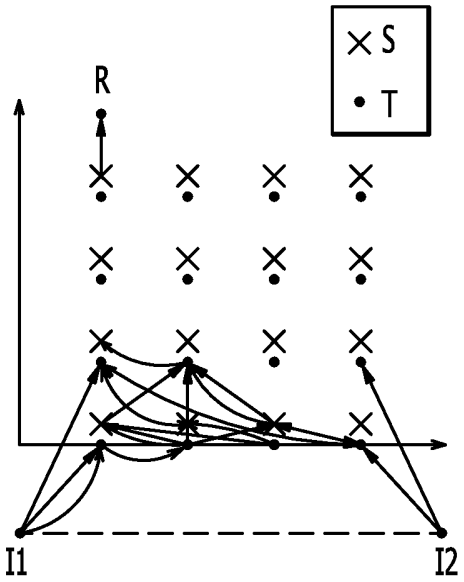


FIG.2

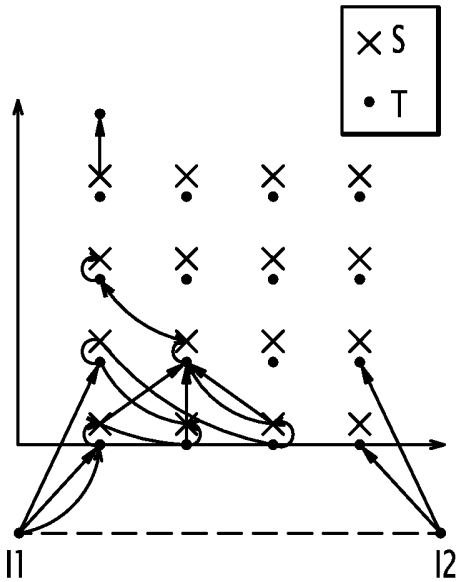


FIG.3

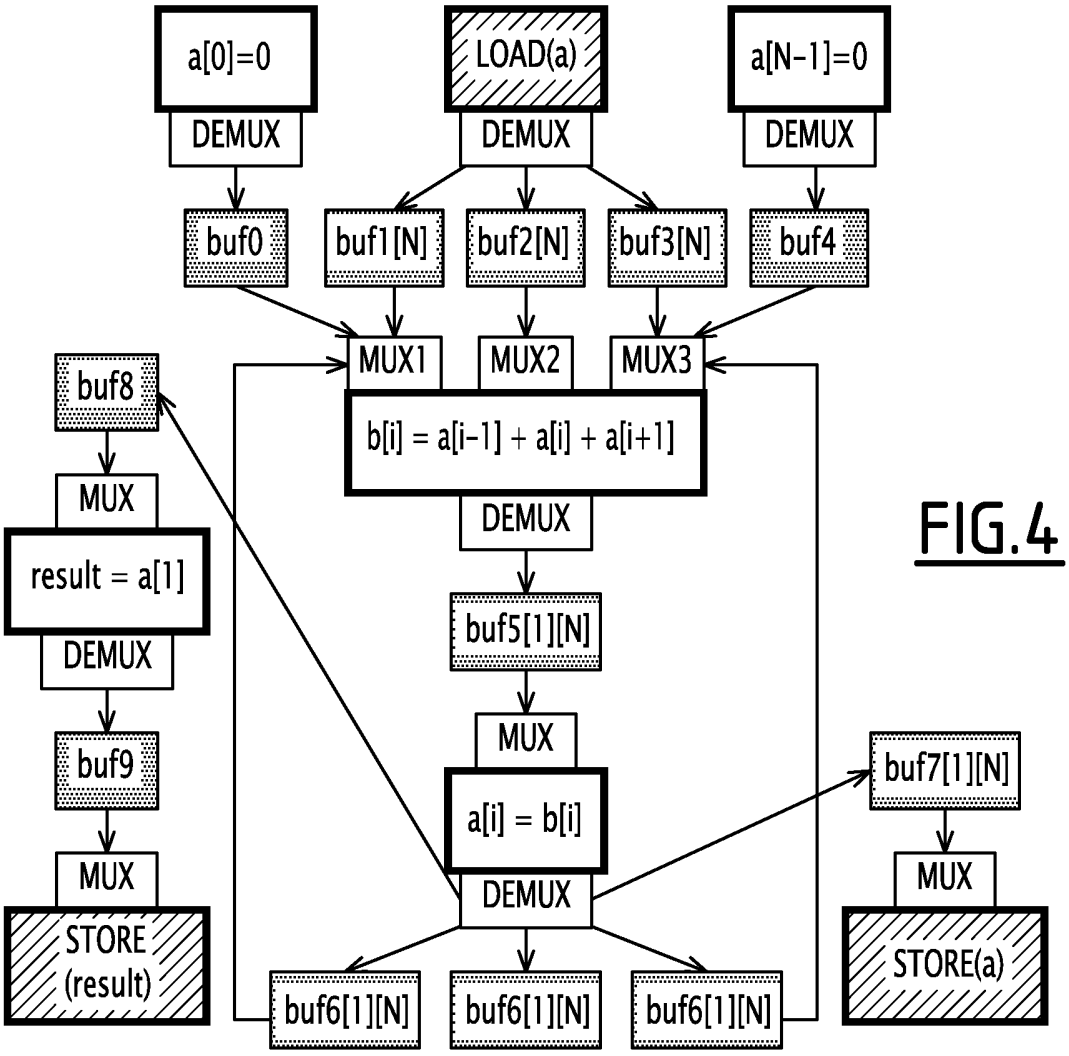

a[0]=0
LOAD(a)
a[N-1]=0
DEMUX
DEMUX
DEMUX
buf0
buf1[N]
buf2[N]
buf3[N]
buf4
MUX1
MUX2
MUX3
buf8
b[i] = a[i-1] + a[i] + a[i+1]
MUX
DEMUX
result = a[1]
buf5[1][N]
DEMUX
MUX
buf9
a[i] = b[i]
buf7[1][N]
MUX
DEMUX
MUX
STORE (result)
STORE(a)
buf6[1][N]
buf6[1][N]
buf6[1][N]

FIG.4

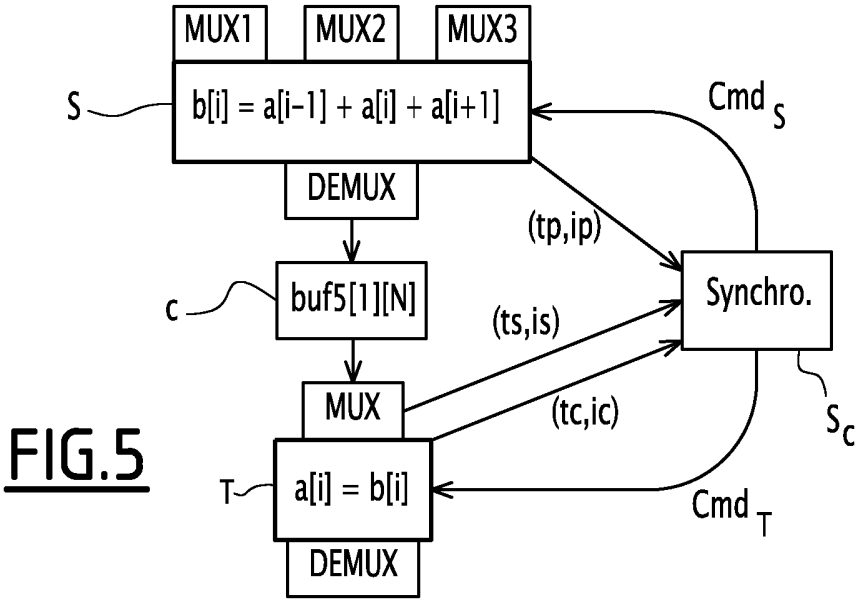

MUX1
MUX2
MUX3
S
b[i] = a[i-1] + a[i] + a[i+1]
Cmd S
DEMUX
(tp,ip)
c
buf5[1][N]
Synchro.
(ts,is)
MUX
(tc,ic)
Sc
T
a[i] = b[i]
Cmd T
DEMUX

FIG.5

RÉPUBLIQUE FRANÇAISE

INSTITUT NATIONAL DE LA PROPRIÉTÉ INDUSTRIELLE

RAPPORT DE RECHERCHE PRÉLIMINAIRE

établi sur la base des dernières revendications déposées avant le commencement de la recherche

N° d'enregistrement national

FA 797769
FR 1453308

DOCUMENTS CONSIDÉRÉS COMME PERTINENTS		Revendication(s) concernée(s)	Classement attribué à l'invention par l'INPI
Catégorie	Citation du document avec indication, en cas de besoin, des parties pertinentes		
X	SVEN VAN HAASTREGT ET AL: "Enabling Automatic Pipeline Utilization Improvement in Polyhedral Process Network Implementations", APPLICATION-SPECIFIC SYSTEMS, ARCHITECTURES AND PROCESSORS (ASAP), 2012 IEEE 23RD INTERNATIONAL CONFERENCE ON, IEEE, 9 juillet 2012 (2012-07-09), pages 173-176, XP032473234, DOI: 10.1109/ASAP.2012.23 ISBN: 978-1-4673-2243-0 * le document en entier *	1-15	G06F9/46
A	MICHAEL BEYER ET AL: "Static Analysis of Run-Time Modes in Synchronous Process Network", 27 juin 2011 (2011-06-27), PERSPECTIVES OF SYSTEMS INFORMATICS, SPRINGER BERLIN HEIDELBERG, BERLIN, HEIDELBERG, PAGE(S) 55 - 67, XP019177337, ISBN: 978-3-642-29708-3 * le document en entier *	1-15	DOMAINES TECHNIQUES RECHERCHÉS (IPC) G06F
A	WO 2010/140883 A2 (VECTOR FABRICS B V [NL]; VAN EIJNDHOVEN JOS [NL]; KAMPS TOMMY [NL]; KA) 9 décembre 2010 (2010-12-09) * le document en entier *	1-15	

Date d'achèvement de la recherche	Examinateur
28 janvier 2015	Radev, Boyan

CATÉGORIE DES DOCUMENTS CITÉS

X : particulièrement pertinent à lui seul
Y : particulièrement pertinent en combinaison avec un autre document de la même catégorie
A : arrière-plan technologique
O : divulgation non-écrite
P : document intercalaire

T : théorie ou principe à la base de l'invention
E : document de brevet bénéficiant d'une date antérieure à la date de dépôt et qui n'a été publié qu'à cette date de dépôt ou qu'à une date postérieure.
D : cité dans la demande
L : cité pour d'autres raisons
.....
& : membre de la même famille, document correspondant

EPO FORM 1503 12.99 (P04C14)

ANNEXE AU RAPPORT DE RECHERCHE PRÉLIMINAIRE RELATIF A LA DEMANDE DE BREVET FRANÇAIS NO. FR 1453308 FA 797769

La présente annexe indique les membres de la famille de brevets relatifs aux documents brevets cités dans le rapport de recherche préliminaire visé ci-dessus.
Les dits membres sont contenus au fichier informatique de l'Office européen des brevets à la date du 28-01-2015
Les renseignements fournis sont donnés à titre indicatif et n'engagent pas la responsabilité de l'Office européen des brevets, ni de l'Administration française

Document brevet cité au rapport de recherche	Date de publication	Membre(s) de la famille de brevet(s)	Date de publication
WO 2010140883 A2	09-12-2010	EP 2438545 A2	11-04-2012
		US 2012144376 A1	07-06-2012
		WO 2010140883 A2	09-12-2010

EPO FORM P0465