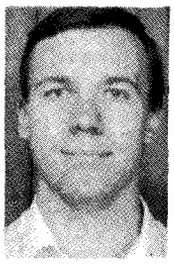
Formalisation en occam et simulation sur un réseau de Transputers d'un processeur systolique de traitement d'images

Description in occam and simulation on a Transputer network of a systolic processor for image processing



Alain PIRSON

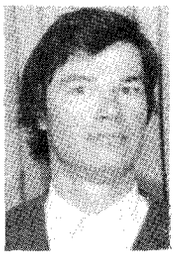
C.E.A. D.LETI/DSYS/SETIA, Centre d'études Nucléaires, 38041 GRENOBLE CEDEX.

Alain Pirson est né en 1963 à Namur en Belgique. Il obtient en 1986 le grade d'Ingénieur Civil Électricien à l'Université Catholique de Louvain (Belgique). Il mène ensuite une activité de recherche à la division LETI du Commissariat à l'Énergie Atomique dans le cadre d'un Plan de Stimulation des Coopérations et des Échanges Scientifiques et Techniques Européens. Spécialisé dans les architectures parallèles pour le traitement d'images, il prépare actuellement une thèse de Docteur-Ingénieur de l'INPG (Institut National Polytechnique de Grenoble).

Dominique DAVID

C.E.A. D.LETI/DSYS/SETIA, Centre d'études Nucléaires, 38041 GRENOBLE CEDEX.

Diplômé de l'École Supérieure d'Électricité, Dominique David obtient son doctorat d'ingénieur en traitement du signal à Grenoble en 1980. Il travaille à la division LETI du Commissariat à l'Énergie Atomique depuis 1983 dans le domaine de la vision par ordinateur et des architectures pour le traitement d'images. Actuellement, il s'occupe d'une part de l'étude d'algorithmes pour les problèmes d'inspection et de contrôle, et d'autre part du développement de circuits intégrés mettant en œuvre ces algorithmes.



Jean-Luc JACQUOT

C.E.A. D.LETIT/DSYS/SETIA, Centre d'études Nucléaires, 38041 GRENOBLE CEDEX.

Diplômé de l'École Nationale Supérieure d'Aéronautique et de l'Espace en 1985, Jean-Luc Jacquot développe actuellement à la division LETI du Commissariat à l'Énergie Atomique la partie traitement d'images d'une machine d'inspection automatique de circuits intégrés.

Thierry COURT

I2S, 239, rue du Jardin Public, 33041 BORDEAUX CEDEX.

Thierry Court a obtenu en 1984 le diplôme d'Ingénieur électronicien ICPI, en 1985 le DEA d'informatique de l'Université de St-Etienne. Il a étudié au DLETI (CENG, Grenoble) de 1986 à 1989 des architectures parallèles spécialisées pour le traitement d'images et il poursuit cette activité dans le cadre de la société I2S (Bordeaux) depuis novembre 1989.

RÉSUMÉ

L'inconvénient majeur des algorithmes de voisinage (filtrage linéaire, morphologique) réside dans le coût associé à leur mise en œuvre, principalement dans le domaine du traitement d'images où il croît avec le carré de la taille du voisinage. Il a toutefois été montré dans PIR [1] que de nombreux algorithmes couramment utilisés pouvaient être décrits sous la forme d'une cascade de cellules élémentaires particulières. Le caractère hautement intégrable de ces cellules nous a amené à concevoir un processeur systolique de voisinage basé sur le principe de la synthèse par cascades de cellules.

Le passage d'une représentation fonctionnelle (graphe de fluence, modèle mathématique) à une structure matérielle soulève un certain nombre de problèmes. En outre, le coût lié à la réalisation d'un circuit intégré exige une extrême prudence, imposant généralement le recours à de nombreuses simulations.

Le présent papier fournit un bref aperçu de la méthode de synthèse par cascades de cellules. Il montre ensuite la puissance du langage occam comme outil de formalisation appliquant celui-ci à la modélisation des cascades de cellules. La compilation du modèle au moyen d'outils adéquats fournit un code directement exécutable par un réseau de Transputers, ce qui conduit à une simulation performante. La hiérarchie incluse dans le langage permet une décomposition en niveaux d'abstraction de plus en plus fins, en vue d'ajuster la précision du modèle au type de simulation désiré.

MOTS-CLÉS : Occam, réseau de Transputers, simulation, processeur systolique, traitement d'images.

SUMMARY

The most significant drawback in using neighbourhood processing (linear filtering, morphology) is the cost of its implementation, particularly in the area of image processing where the cost grows in proportion to the kernel size squared. However, in PIR [1] *it was shown that a large amount of classical algorithms could be described as a cascade of peculiar elementary cells. Being highly integrable, these cells brought about the design of a neighbourhood systolic processor.*

Converting a functional description (e.g. flow graph, mathematical model) into a hardware structure raises several technical problems. In addition, the cost of designing an integrated circuit requires special care, making it necessary to perform many simulations.

The first part of this paper gives a short description of the decomposition method. It then goes on to describe the power of occam as a design formalism, applying it to the cascaded cells modelisation. Furthermore, the obtained model can be compiled into a code directly loadable into a Transputer network, resulting in a performant simulation. The hierarchy of occam allows a description at different levels of abstraction, in order to fit the precision of the model to the desired type of simulation.

KEY WORDS : *Occam, Transputer network, simulation, systolic processor, image processing.*

1. Introduction

La conception d'un circuit intégré est une opération relativement complexe. Partant de la représentation fonctionnelle d'un ensemble de comportements logiques, elle aboutit à une structure physique matérialisée par un composant. Elle fait appel en cela à de nombreux niveaux de description (blocs fonctionnels, macro-cellules, portes logiques...).

L'utilisation d'outils de conception de plus en plus évolués réduit considérablement le risque d'erreurs lors du passage d'un niveau de description à un autre. Il demeure toutefois une étape où la machine n'a pas encore supplanté le concepteur. Il s'agit de la conversion d'une représentation fonctionnelle (en terme de comportements logiques) en une structure matérielle. Cette étape demande en effet méthode, habileté et expérience et résulte de nombreux compromis. Opération complexe, elle comporte le risque d'une dérive par rapport au fonctionnement souhaité.

Heureusement, l'opération inverse, consistant à remonter au comportement à partir d'une structure est parfaitement accessible aux machines, par le biais de simulations. Pour ce, il est nécessaire de disposer d'un outil de formalisation rigoureux permettant de décrire une structure matérielle en termes compréhensibles par une machine de simulation. Ce formalisme doit rendre compte de la simultanéité de fonctionnement des éléments de la structure ainsi que de leur influence mutuelle. Le langage occam paraît bien adapté à cet égard. L'objet du présent papier est de l'illustrer au travers d'un exemple concret : la réalisation d'un processeur de voisinage basé sur le principe de la synthèse par cascades de cellules.

2. Synthèse par cascades de cellules

Nous avons montré dans les articles PIR [1], JAC [2] et DAV [3] que de nombreux filtres de traitement d'images, tant linéaires que morphologiques, pouvaient être synthétisés au moyen d'une cascade de cellules appartenant aux deux classes suivantes :

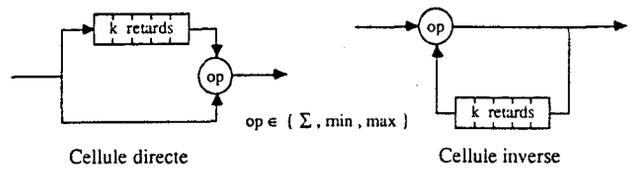


Fig. 1.

Les quelques exemples qui suivent illustrent brièvement les principes de cette approche.

2.1. Filtrage uniforme (ou moyenne glissante)

Le filtrage uniforme d'un signal monodimensionnel x_n consiste à évaluer la moyenne des m échantillons voisins de x_n :

$$y_n = \sum_{k=0}^{m-1} x_{n-k} \cdot$$

Nous avons volontairement supprimé le facteur d'échelle qui correspond à une adaptation de la dynamique du signal sortant, opération réalisable en fin de traitement.

Appliquons la transformée en Z à cette expression :

$$Y = X \sum_{k=0}^{m-1} z^{-k} = X \frac{1 - z^{-m}}{1 - z^{-1}} \cdot$$

Le résultat obtenu peut être interprété comme un système linéaire où x et y sont respectivement les signaux d'entrée et de sortie et dont la fonction de transfert est $(1 - z^{-m})/(1 - z^{-1})$:

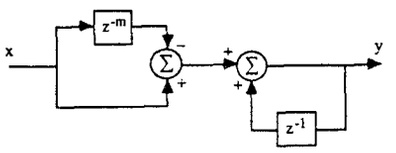


Fig. 2. — Graphe du filtre uniforme.

Il est facile d'étendre cette approche au traitement d'images. Par simple balayage vidéo, les images se réduisent à des signaux et les opérations bi-dimensionnelles à des opérations monodimensionnelles. Ainsi, le filtrage uniforme de taille 5 correspond à la convolution de l'image initiale par le masque séparable ([1]) suivant :

$$\begin{bmatrix} 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \end{bmatrix} = \begin{bmatrix} 1 \\ 1 \\ 1 \\ 1 \\ 1 \end{bmatrix} [1\ 1\ 1\ 1\ 1] .$$

Appelons N, le nombre de pixels que comporte une ligne de l'image à traiter. L'équation du système linéaire 1D correspondant s'écrit dès lors

$$\begin{aligned} Y &= (1 + z^{-1} + z^{-2} + z^{-3} + z^{-4}) \cdot \\ &\quad (1 + z^{-N} + z^{-2N} + z^{-3N} + z^{-4N}) \cdot X \\ &= [(1 - z^{-5})/(1 - z^{-1})] \cdot \\ &\quad [(1 + z^{-5N})/(1 - z^{-N})] \cdot X . \end{aligned}$$

Il y correspond le diagramme suivant, où les puissances de z sont représentées par des retards :

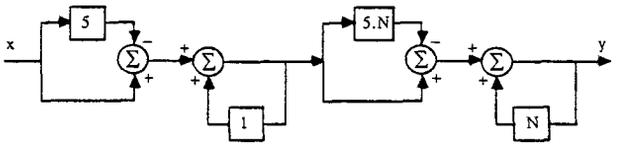


Fig.3. — Diagramme du filtre uniforme 5 x 5 sous forme d'une cascade de cellules.

2.2. Gradient vertical de Sobel

Ce filtre est principalement utilisé pour déterminer les contours dans une image.

— Masque du gradient vertical 3 x 3 de Sobel :

$$\begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix} = \begin{bmatrix} -1 \\ 0 \\ 1 \end{bmatrix} * [1\ 2\ 1] .$$

L'équation du filtre et le diagramme correspondant sont les suivants :

$$\begin{aligned} Y &= (1 + 2 . z^{-1} + z^{-2}) . (1 - z^{-2.N}) . X = \\ &= (1 + z^{-1})^2 . (1 - z^{-2.N}) . X . \end{aligned}$$

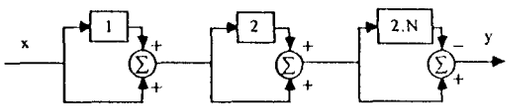


Fig. 4. — Diagramme du gradient vertical de Sobel sous forme d'une cascade de cellules.

A titre d'exemple, le filtre dont les coefficients sont présentés ci-après correspond à un opérateur de gradient horizontal 9 x 9. Il a été mis au point à partir de la théorie de Canny et s'inscrit dans le cadre des extracteurs de contours à noyau large (JAC [2]). La cascade de cellules s'obtient par factorisation des polynômes en Z

$$[1\ 4\ 8\ 12\ 14\ 12\ 8\ 4\ 1]^T * [1\ 5\ 10\ 9\ 0\ -9\ -10\ -5\ -1]$$

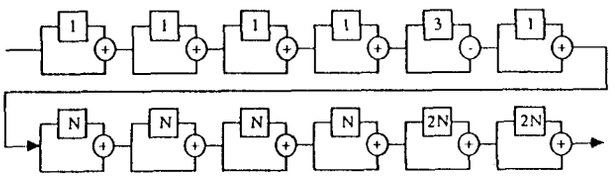


Fig. 5. — Diagramme d'un opérateur de gradient horizontal à noyau large.

2.3. Érosion et dilatation

Ce mode de décomposition est également applicable à certains filtres non linéaires couramment utilisés par la morphologie en niveaux de gris.

Considérons l'érosion d'une image par un élément structurant de support B, laquelle s'écrit :

$$y_{m,n} = \min_{k,\, l \in B} \{x_{m-k,\, n-l}\}$$

([1]) Un filtre 2D est dit séparable s'il peut se ramener à deux filtres 1D ; ce sera le cas de tous les filtres que nous envisagerons.

En fonction de la nature de l'élément structurant, cette opération devient séparable et se réduit à deux (voire plus) érosions monodimensionnelles :

$$y_{m,n} = \min_{k \in B1} \left\{ \min_{l \in B2} \{x_{m-k,n-l}\} \right\}$$

avec B1 et B2 supports d'éléments structurants 1D tels que la dilatation de B1 par B2 donne B. Par ailleurs, il a été montré dans PIR [1] que l'érosion et la dilatation mono-dimensionnelles sont synthétisables par cascades de cellules, et que le nombre de cellules requises correspond au $\log_2$ de la taille de l'élément structurant. Ainsi l'érosion d'une image au moyen d'un élément structurant carré de côté 4 nécessite 2 $\log_2$ 4 soit 4 cellules :

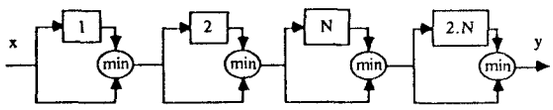


Fig. 6. — Diagramme de l'érosion 4 x 4 sous forme d'une cascade de cellules.

Pour obtenir de plus amples détails sur la synthèse par cascade de cellules, il est conseillé de se référer à la bibliographie attenante.

Une telle méthode de décomposition permet de réduire considérablement la complexité et le nombre d'opérations nécessaires. Ainsi, le filtrage linéaire ne requiert aucune multiplication. D'une manière générale, les seules opérations impliquées sont l'addition, la soustraction, le minimum et le maximum, soit des opérations peu coûteuses. Par ailleurs, les cellules mises en œuvre se caractérisent par une extrême simplicité et une grande régularité. Elles n'exigent qu'un nombre limité de chemins de données et autorisent le traitement des informations à la volée (data flow). Ces divers éléments montrent que les cascades de cellules se prêtent à une implantation sur silicium.

La représentation des algorithmes de voisinage sous forme de cascades de cellules ne présente un véritable intérêt que si elle débouche sur un circuit intégré. Dans cette optique nous avons entrepris la conception d'un processeur systolique de traitement de voisinage.

3. Passage du modèle mathématique à la structure matérielle

La théorie mathématique associée à la méthode de synthèse fournit une représentation formelle des cellules et de leurs éléments constitutifs. Le passage à une structure matérielle s'effectue par le remplacement des objets du modèle mathématique par des composants physiques. Ainsi, les opérateurs deviennent des unités arithmétiques et logiques et les retards des registres à décalage. Il est évident que les hypothèses sur lesquelles repose le modèle mathématique ne s'appliquent pas nécessairement aux composants physiques.

Ainsi :

— La précision des calculs supposée infinie dans le modèle est liée à la largeur des unités arithmétiques et logiques et des chemins de données. Elle est donc finie et peut introduire des erreurs d'arrondis.

— Les registres à décalage sont des éléments de mémorisation. Ils contiennent à ce tire une valeur dès leur mise sous tension. L'ensemble des valeurs présentes constitue des conditions initiales, lesquelles produisent des réponses transitoires qui se superposent aux réponses naturelles des cellules excitées par un signal donné. Le signal obtenu en sortie d'une cascade peut s'avérer totalement erroné.

— La transformation d'une image en un signal par balayage conduit à la perte de la notion de début et de fin de ligne. Ainsi, des pixels n'ayant aucune proximité dans l'image se retrouvent voisins dans le signal. Les traitements au moyen de cascades de cellules introduisent dès lors des effets de bord dont il est indispensable de tenir compte.

Ces phénomènes naissant lors du passage à une structure matérielle doivent être analysés soigneusement. Il est par ailleurs nécessaire de valider la synthèse par cascades de cellules au moyen d'images test afin de vérifier qu'une telle opération est licite. Le coût élevé lié à la réalisation d'un circuit impose en effet un maximum de précautions.

Une des méthodes les plus efficaces à cet égard est la simulation comportementale. Elle consiste à modéliser de manière plus ou moins fidèle le comportement des éléments constitutifs de la structure matérielle. La simulation comportementale réalise ainsi le lien entre le modèle mathématique et le système physique. La complexité du modèle comportemental et la charge de calcul qu'il représente pour un simulateur sont fonction du niveau de précision exigé lors de la simulation. Le choix d'un modèle est dicté par le type de comportement que l'on désire mettre en évidence.

La simulation comportementale est un complément idéal à la réflexion théorique, confirmant ou réfutant les hypothèses avancées. Offrant une approche nouvelle, elle peut dans certains cas faire ressortir des comportements imprévus. En résumé, elle constitue un outil indispensable au concepteur de systèmes.

La modélisation de comportements nécessite un outil de formalisation permettant d'exprimer ceux-ci en termes simples. Il semble à ce titre qu'un langage parallèle soit particulièrement bien adapté. En effet, il permet d'une part de tenir compte du parallélisme naturel présent dans les cascades de cellules et d'autre part, en temps que langage de programmation, de fournir un texte utilisable par une machine de simulation.

4. Choix d'un langage parallèle

La programmation parallèle pose deux problèmes dont un langage doit tenir en compte :

— le découpage d'un programme en tâches concurrentes partageant des informations,

— la répartition des tâches entre les diverses unités de traitement de la machine.

Un langage approche ces problèmes de manière très élégante : le langage occam. Issu de CSP (Communicating Sequential Processes), il repose sur un modèle de parallélisme semblable, fondé sur la notion de processus. D'après ce modèle, un système parallèle est composée d'un ensemble de processus concurrents dialoguant entre eux par l'échange de messages au travers de canaux.

Chaque processus constitue une entité autonome capable d'effectuer une série d'actions sur des objets qui lui sont propres et dispose à cet égard de son système de séquencement. Le modèle exclut toute forme de partage. Un objet appartient à un seul processus et ne peut être commun à plusieurs.

L'échange d'informations entre processus concurrents s'effectue au travers de canaux unidirectionnels reliant un processus émetteur et un processus récepteur. Il est régi par un mécanisme précis basé sur le concept de rendez-vous. D'après ce mécanisme, le passage d'une information d'un processus émetteur à un processus récepteur n'a effectivement lieu que lorsque les processus concernés sont simultanément prêts pour l'échange. Dès lors, le premier processus impliqué dans un dialogue est momentanément interrompu dans l'attente de son correspondant. Tout échange provoque donc une resynchronisation des processus. Un tel mécanisme assure une grande sécurité au niveau du partage d'informations. Il est par ailleurs tout à fait indépendant du site d'implantation.

Occam inclut la notion de hiérarchie. Toute action interne à un processus peut elle-même être un ou plusieurs processus.

Le modèle occam du parallélisme favorise une approche rigoureuse et élégante des problèmes liés à la concurrence.

Plus qu'un langage de programmation parallèle, occam est un puissant outil de description. Il permet de représenter au moyen d'un formalisme simple le parallélisme présent dans les applications et se prête dès lors à la simulation comportementale de cascades de cellules.

5. Choix d'une machine de simulation

De très nombreuses machines sont théoriquement susceptibles de supporter des programmes respectant le modèle occam du parallélisme.

Ainsi, un processus peut être exécuté par un ordinateur d'usage général (processeur + mémoire + code exécutable), par un opérateur spécialisé (unité arithmétique et logique + séquenceur), par un automate à états finis (logique séquentielle), en logique combinatoire...

De même, le mécanisme de canal peut être réalisé par un échange au travers d'une mémoire ou d'un registre commun, par un échange au travers d'un lien asynchrone...

Un composant est spécialement conçu pour recevoir les programmes en occam : le Transputer de INMOS. Il s'agit d'un micro-ordinateur composé d'un microprocesseur de type RISC, d'une mémoire interne et de quatre liens série grâce auxquels il se connecte à ses semblables pour former un réseau de géométrie choisie. La philosophie du langage occam est respectée : les processeurs exécutent les processus et les liens matérialisent les canaux, media de dialogue. Outre les instructions nécessaires à l'écriture du texte des processus, le langage occam est pourvu d'instructions permettant la description logique de la topologie du réseau cible, la formalisation de l'association lien-canal et processeur-processus. Un outil de développement a été conçu par INMOS. Cet outil est capable à partir du texte complet d'un programme en occam de générer du code Transputer et de répartir celui-ci entre les divers Transputers d'un réseau en fonction de la description logique de sa topologie.

Le Transputer est un puissant micro-ordinateur d'usage général. Son aptitude à former des réseaux de topologie quelconque fournit un moyen simple de réaliser des machines parallèles très performantes. Le choix du langage occam comme outil de formalisation des cascades de cellules nous a conduit à retenir un système à base de Transputers comme machine de simulation. Un tel choix se justifie d'autant plus que la simulation est une opération lourde, requérant une grande puissance de calcul.

6. Simulation d'une cascade de cellules

La présente section montre comment au moyen du langage occam il est possible d'appréhender simplement la formalisation d'un problème donné.

6.1. Description en occam d'une cascade de cellules

Considérons la cascade correspondant au Laplacien 3×3 et décrivons-la en occam afin de la simuler sur un réseau de Transputers.

Masque du Laplacien 3×3 :

$$\begin{bmatrix} 1 & -2 & 1 \\ -2 & 4 & -2 \\ 1 & -2 & 1 \end{bmatrix} = \begin{bmatrix} 1 \\ -2 \\ 1 \end{bmatrix} * [1 \quad -2 \quad 1]$$

Equation du filtre et le diagramme correspondant :

$$Y = (1 - 2 . z^{-1} + z^{-2}) . (1 - 2 . z^{-N} + z^{-2N}) . X$$
$$= (1 - z^{-1})^2 . (1 - z^{-N})^2 . X$$

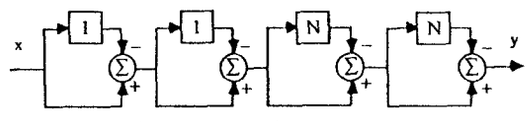


Fig. 7. — Diagramme du Laplacien sous forme d'une cascade de cellules.

La première constatation possible est l'indépendance du fonctionnement des différentes cellules de filtrage. Elles peuvent aisément être associées à des processus concurrents échangeant les résultats de leurs calculs sous la forme de messages.

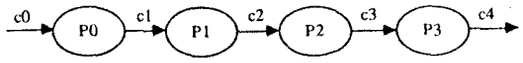


Fig. 8.

Le programme occam décrivant un tel graphe s'écrit :

PAR
$P_0(c_0, c_1)$
$P_1(c_1, c_2)$
$P_2(c_2, c_3)$
$P_3(c_3, c_4)$

signifiant « mise en parallèle des processus P_0, P_1, P_2, P_3 ayant respectivement comme paramètres les canaux (c_0, c_1), (c_1, c_2), (c_2, c_3), (c_3, c_4) ». Il est agréable de remarquer la simplicité de la description.

Étudions à présent en détail le comportement des processus P_i.

Nous avons recensé deux types de cellules : les cellules directes et les cellules inverses. Chaque cellule d'un type donné est capable d'effectuer une des opérations suivantes (+, −, min, max). Cela signifie huit types de comportements différents. Il est donc nécessaire de décrire huit types de processus différents. Chaque processus décrit recevra comme paramètre le nombre de retards que la cellule modélisée comporte.

A titre d'exemple, analysons une cellule directe de soustraction. Une telle cellule peut être considérée comme composée de trois parties (*fig.* 9).

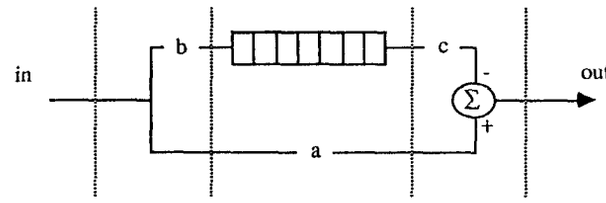


Fig. 9

Les trois parties fonctionnant simultanément peuvent être décrites par trois processus concurrents reliés par des canaux.

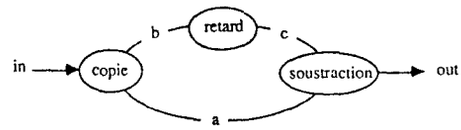


Fig. 10.

Il y correspond le texte occam suivant :

```
PROC SOUSTRACTION.DIRECTE (CHAN OF INT in, out, VAL OF INT nombre.de.retards)
  — déclare le processus SOUSTRACTION.DIRECTE ayant pour paramètres les canaux d'entiers in et out
    et la valeur entière nombre.de.retards
  CHAN OF INT a, b, c :  — déclare les canaux d'entiers a, b et c
  PAR                    — mise en parallèle des processus qui suivent.
    COPIE (in, a, b)
    RETARD (b, c, nombre.de.retards)
    SOUSTRACTION (a, c, out)
:
```

Le premier processus appelé COPIE se contente de lire les données sur le canal entrant *in* et de les envoyer simultanément vers les deux autres processus au travers des canaux *a* et *b*. De manière plus précise, il effectue une boucle dans laquelle il commence par lire une donnée sur le canal *in* pour ensuite l'envoyer simultanément sur les canaux *a* et *b*. Un tel comportement s'écrit en occam :

```
PROC COPIE (CHAN OF INT in a, b)
  — déclare le processus COPIE ayant pour paramètres les canaux d'entiers in, a et b.
  INT x :              — déclarer une variable entière x
  WHILE TRUE           — répéter indéfiniment
    SEQ                  — de manière séquentielle
      in ? x               — lire une valeur dans le canal in et la mettre dans x
      PAR                  — ensuite effectuer simultanément
        a ! x                — l'envoi de la valeur de x dans le canal a
        b ! x                — et dans le canal b
:
```

Remarque sur l'interprétation du texte précédent :

La notion de processus est clairement formalisée en occam : un processus est une suite d'instructions ou de processus précédée d'un constructeur indiquant leur ordre d'exécution. Les constructeurs dont nous aurons besoin ici sont le **SEQ** signifiant exécution séquentielle des éléments sur lesquels il porte, le **PAR** signifiant exécution simultanée, le **WHILE** indiquant la répétition du processus auquel il s'appli-

que. L'indentation fixe les limites de la portée des constructeurs. Toutes les instructions ou processus situés au même niveau d'indentation dépendent du même constructeur. Le niveau d'indentation d'un processus se mesure par celui de son constructeur de tête.

Le second processus appelé RETARD introduit un certain nombre de retards, ce nombre étant fixé par le paramètre *nombre.de.retards*. Il s'agit en fait d'un registre à décalage que nous pouvons représenter sous la forme d'une cascade de bascules. Celles-ci fonctionnant toutes simultanément, elles peuvent être décrites par une chaîne de processus concurrents reliés par des canaux (*fig.* 9).

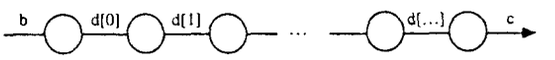


Fig. 11.

Une telle chaîne de processus s'écrit en occam

```
PROC RETARD (CHAN OF INT b, c, VAL OF INT nombre.de.retards)
  — déclare le processus RETARD ayant pour paramètres les canaux d'entiers b et c et la valeur entière
    nombre.de.retards.
  [N.MAX] CHAN OF INT d:          — déclare un vecteur d de N.MAX canaux
  PAR                             — mise en parallèle
    bascule (b, d[0]              — d'un processus bascule
    PAR i=0 FOR nombre.de.retard-2 — d'une suite de (nombre.de.retard-2)
      bascule (d[i], d[i+1])        processus bascule en parallèle
    bascule (d[nombre.de.retard-2], c — d'un processus bascule
:
```

Ce texte signifie mise en parallèle de plusieurs processus identiques possédant le même nom *bascule,* mais dialoguant au travers de canaux différents. Les plus astucieux auront remarqué que le processus RETARD nécessite une légère modification dans le cas où le nombre de retards est inférieur à 2 (suppression d'un appel du processus bascule).

Le processus *bascule* modélise le comportement de la bascule classique :

```
PROC bascule (CHAN OF INT in, out)
  — processus bascule ayant comme paramètres les canaux de type entier in et out
  INT x, y:              — Déclarer les variables entières x et y
  SEQ                    — De manière séquentielle
    y := 0               — remettre la bascule à zéro
    WHILE TRUE           — répéter indéfiniment
      SEQ                — de manière séquentielle les opérations suivantes :
        PAR              — d'abord simultanément
          in ? x         — lire une valeur dans le canal in et la mettre dans x
          out ? y        — envoyer dans le canal out le contenu de y
        y := x           — ensuite actualiser la valeur de y
:
```

Cette modélisation du processus RETARD a l'avantage d'être conceptuellement très simple. Bien adaptée pour un petit nombre de retards, elle s'avère inefficace lorsque celui-ci augmente. En effet, son coût mesuré en nombre de processus concurrents croît linéairement avec le nombre de retards désirés, ce qui pour des retards importants (plusieurs lignes d'une image) conduit à des quantités démesurées pénalisant inutilement la machine de simulation.

Une autre version du processus RETARD, simulant la gestion d'une mémoire tampon contenant les derniers échantillons du signal entrant, a été développée. Cette version est considérablement plus performante, son coût étant indépendant du nombre de retards.

Le troisième processus appelé SOUSTRACTION modélise le comportement d'un soustracteur. Il consiste à lire une valeur sur chacun des canaux entrant et à envoyer sur le canal sortant le résultat de la soustraction des deux valeurs acquises.

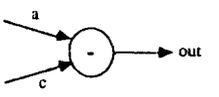


Fig. 12.

```
PROC SOUSTRACTION (CHAN OF INT a, c, out)
— processus soustraction ayant comme paramètres les canaux de type entier a, c et out
  WHILE TRUE           — Répéter indéfiniment
    INT x, y:          — déclarer les variables entières x et y ;
```

```
SEQ                    — de manière séquentielle
  PAR                  — d'abord simultanément
    a ? x              — lire une valeur dans le canal a et la mettre dans x
    c ? y              — lire une valeur dans le canal c et la mettre dans y
    out ! x – y        — ensuite envoyer dans le canal out le résultat x – y
:
```

De la même manière, un processus peut être écrit pour les autres opérations possibles (addition, minimum, maximum).

Le texte final du programme est obtenu par association des divers processus décrits et de deux processus supplémentaires permettant l'un de charger la cascade au moyen d'une image test et l'autre de récupérer et de visualiser l'image résultat.

6.2. Implantation sur réseau de Transputers

Un ensemble de processus concurrents au sens d'occam s'exécutent indifféremment sur un ou plusieurs Transputers.

Dans le premier cas, il s'agit d'une concurrence émulée au moyen d'un mécanisme de « temps partagé ». Un seul processus s'exécute à un moment donné, les autres étant placé dans une file d'attente. L'ordonnancement des processus est géré au sein du Transputer par un système micro-codé indépendant du processeur central, afin d'assurer à ce dernier une efficacité maximale. L'échange de messages par canaux est émulé via la mémoire du Transputer.

Dans le cas d'une répartition des processus concurrents entre les Transputers d'un réseau, le parallélisme est réel. Chaque Transputer exécute son processus et dialogue avec ses voisins au travers de ses liens.

Les deux modes d'exécutions peuvent très bien coexister, chaque Transputer d'un réseau exécutant son ensemble de processus par concurrence émulée.

La programmation d'un réseau de Transputers s'effectue en deux étapes :

— écriture du programme parallèle (texte des processus)
— implantation sur le réseau (association liens-canaux, processeurs-processus).

L'association liens-canaux définit indirectement la topologie du réseau.

La simulation est réalisée au moyen d'une carte B003 de INMOS, réseau de quatre Transputers disposant chacun de 256 Ko de mémoire. Les Transputers sont des T414-20 (micro-ordinateurs 32-bits dont l'horloge interne fonctionne à 20 MHz, offrant une performance de 10 MIPS chacun). Ce réseau est piloté par un PC au travers d'une carte B004, composée d'un Transputer IMS T414 et de 2 Mo de mémoire.

Le rôle principal de la carte pilote est l'interface entre le monde des Transputers et l'environnement du PC. Le Transputer de cette carte est le seul d'un réseau capable d'accéder au travers du PC à ses périphériques (console, disques, moniteurs...). Il supporte l'outil de développement permettant la programmation d'un réseau de Transputers (compilateur occam, extracteur de code, chargeur du réseau, ...) et sera responsable, dans notre application, de l'exécution des processus de chargement et de déchargement des images test présentes sur le disque du PC.

Le réseau de la carte B003 est configurable par l'utilisateur. La structure linéaire de la chaîne de processus simulant une cascade de cellules nous conduit à adopter une architecture pipe-line.

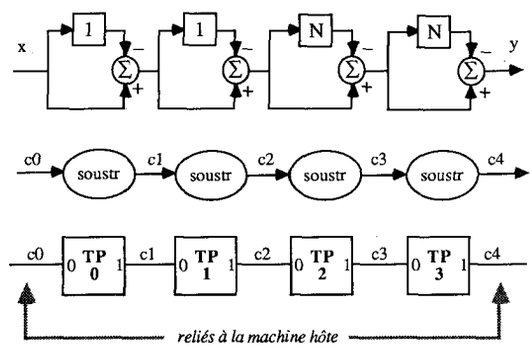


Fig. 13.

Le texte occam correspondant au placement des processus est le suivant :

```
... SC SOUSTRACTION.DIRECTE (in, out, nombre.de.retards)

  — code du processus soustraction.directe
CHAN OF INT c0, c1, c2, c3, c4 :     — déclare les canaux reliant les processus
PLACED PAR                           — placer en parallèle
  PROCESSOR 0                          sur le processeur 0
    PLACE c0 AT input.link[0] :        — placer le canal c0 sur le lien entrant 0
    PLACE c1 AT output.link[1] :       — placer le canal c1 sur le lien sortant 1
    SOUSTRACTION.DIRECTE (c0, c1, 1)   — exécuter le processus soustraction.directe
  PROCESSOR 1                        — sur le processeur 1
    PLACE c1 AT input.link[0]          — placer le canal c1 sur le lien entrant 0
    PLACE c2 AT output.link[1]         — placer le canal c2 sur le lien sortant 1
    SOUSTRACTION.DIRECTE (c1, c2, 1)   — exécuter le processus soustraction.directe
```

```
PROCESSOR 2                              — sur le processeur 2
  PLACE c2 AT input.link[0]                — placer le canal c2 sur le lien entrant 0
  PLACE c3 AT output.link[1]               — placer le canal c3 sur le lien sortant 1
  SOUSTRACTION.DIRECTE (c2, c3, N)         — exécuter le processus soustraction.directe
PROCESSOR 3                              — sur le processur 3
  PLACE c3 AT input.link[0] :              — placer le canal c3 sur le lien entrant 0
  PLACE c4 AT output.link[1] :             — placer le canal c4 sur le lien sortant 1
  SOUSTRACTION.DIRECTE (c3, c4, N)         — exécuter le processus soustraction.directe.
```

L'outil de développement est capable d'interpréter ce texte en vue d'une implantation sur le réseau de Transputers.

7. Résultats des simulations

Diverses simulations ont été effectuées au moyen d'un ensemble d'images test acquises en milieu industriel. Dans un premier temps, elles ont confirmé la validité de la synthèse par cascade de cellules en fournissant des résultats quasi identiques à ceux obtenus par filtrage bi-dimensionnel standard. Les seules différences se situent au niveau des bords d'image où la notion de voisinage n'est pas clairement définie. Les valeurs obtenues dans ces zones sont dépourvues de signification, mais ne nuisent pas à la qualité des images résultat.

L'étude théorique de l'évolution de la dynamique d'un signal numérique en fonction des traitements qu'il subit est très complexe et conduit souvent à des valeurs excessives, étant menée dans les conditions les plus défavorables (valeurs au pire cas). Des simulations au moyen d'images réelles, nous ont permis de quantifier de manière empirique cette évolution en étudiant la dégradation des résultats en fonction de la réduction artificielle de la dynamique. Nous nous sommes aperçus que pour la plupart des traitements, les limites pratiques étaient bien inférieures aux limites théoriques. Toutefois, certaines cascades de cellules provoquent une augmentation du niveau moyen du signal traité. Il est dès lors nécessaire de recaler celui-ci de temps à autre surtout lorsque l'on enchaîne de nombreux traitements. C'est pourquoi il est conseillé de placer une table de transcodage (LUT) en sortie du processeur. Celle-ci permettra en outre d'effectuer quelques post-traitements comme l'élévation au carré (utile pour la variance) ou la valeur absolue.

La théorie des filtres nous apprend que la cellule inverse est instable, car tous ses pôles sont situés sur le cercle unité. Cette cellule est néanmoins utilisable pour autant qu'elle soit accompagnée d'une cellule directe capable de « neutraliser » son instabilité. Par exemple $1/(1-z^{-1})$ est instable alors que $(1-z^{-m})/(1-z^{-1})$ ne l'est pas. Les simulations nous ont montré que les positions relatives des cellules dans la cascade revêtent également une importance, alors que la théorie reste muette à ce sujet. Ainsi, il est indispensable de placer *en tête* d'une cellule inverse la cellule directe qui la neutralise, sans quoi la cellule inverse diverge. Par contre, si l'ordre des cellules est correctement respecté, l'instabilité locale est compensée par les cellules amont et seul le traitement global provoque une modification du niveau moyen du signal.

Un autre enseignement concerne l'influence des valeurs contenues dans les registres du processeur avant le début des calculs. Autant leur influence sur les cellules directes est nulle, sauf sur les premiers résultats (effets de bord), autant elle est catastrophique sur les cellules inverses, conduisant à des résultats erronés et éventuellement à des dépassements de capacité. Il est dès lors fondamental de remettre à *zéro* tous les registres des cellules inverses avant chaque traitement. Il faut savoir qu'une erreur survenue en cours de traitement sur une cellule inverse se perpétue jusqu'à la fin du traitement et fausse tous les résultats. Il convient donc d'être prudent.

Bien que la rapidité de traitement ne constitue pas notre motivation principale puisqu'il s'agit de simulations, des temps de réponse courts permettent un travail interactif, plus efficace et plus captivant. Dans ce contexte, les performances et la flexibilité d'un réseau de Transputers en font un outil bien adapté. A titre d'exemple, le traitement d'une image test (composée de 512×512 pixels) par la cascade présentée ci-dessus nécessite plus ou moins 7 secondes, durée tout à fait acceptable dans le cadre de simulations. Le processeur de voisinage dont nous parlerons plus loin réalise le même traitement en 40 ms, mais il ne s'agit plus de simulations.

Une famille de nouveaux opérateurs de gradient a été mise au point en relation avec la théorie de Canny. Ces opérateurs permettent de réaliser des extracteurs de contours performants (JAC [2]). De taille croissante, ils sont tous synthétisables par cascades de cellules. Leur simulation a permis de valider ces nouveaux filtres sur des images réelles.

La méthode de simulation basée sur une modélisation en occam et une implantation sur réseau de Transputers offre la possibilité d'intégrer les opérateurs simulés dans un environnement plus vaste. Il est en effet possible de simuler la suite des opérations effectuées sur l'image sortant d'une cascade de cellules afin d'obtenir une vision globale de la chaîne de traitement. A cet égard, les opérateurs de gradients simulés se sont vu adjoindre des processus de seuillage, de normalisation, d'extraction de maxima locaux, (...), afin d'obtenir des contours de qualité lors de l'extraction à partir des images test. On peut de la même manière simuler une application complète, depuis l'acquisition des images jusqu'à la prise de décision.

Le processeur de voisinage appelé INP20 a été réalisé. Il s'agit d'un processeur systolique composé de six cellules élémentaires en cascade (*fig.* 14).

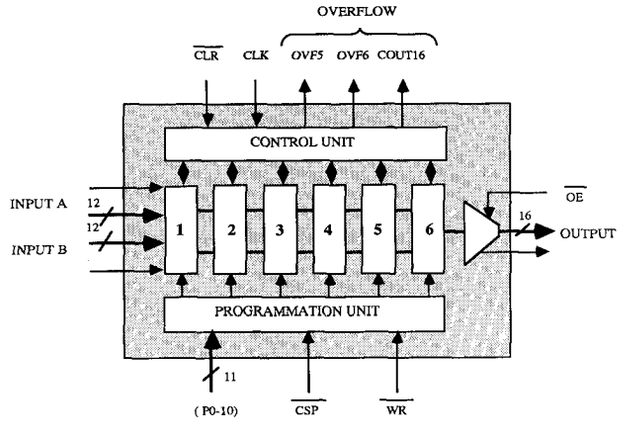


Fig. 14. — Schéma de l'INP20.

Chaque cellule physique de processeur est configurable en une cellule directe ou inverse. L'opération arithmétique ou logique réalisée, de même que le nombre de retards sont programmables. Des chemins de données supplémentaires ont été ajoutés afin de pouvoir effectuer des opérations inter-images et de permettre des combinaisons plus complexes de traitements (*fig.* 15).

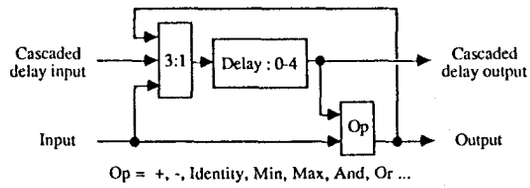


Fig. 15. — Schéma d'une cellule élémentaire de l'INP20.

L'utilisation de la séparabilité et l'introduction de modes de balayages autres que le balayage vidéo nous ont permis de réduire la taille des éléments de mémorisation présents dans chaque cellule. Ainsi, un retard d'une ligne se réduit à un retard d'un pixel lorsque l'image est balayée suivant les colonnes. Il devient dès lors nécessaire d'utiliser deux INP20 et un tampon si l'on désire traiter à la volée le signal vidéo numérisé. Le recours à une mémoire image permet d'obtenir les mêmes performances avec un seul INP20. En outre, cette solution offre la possibilité d'enchaîner plusieurs traitements ou de ne traiter que les zones utiles d'une image.

Chaque INP20 est capable d'effectuer à une cadence typique de 20 millions de pixels par seconde tous les algorithmes de la liste suivante :

— Érosion et dilatation d'images comportant 12 bits de niveaux de gris, avec un élément structurant de taille allant jusqu'à 20.

— Moyenne et variance locales sur des voisinages de taille allant jusqu'à 20.

— Extrema locaux (minimum, maximum) sur des voisinages de taille allant jusqu'à 20.

— Tous les filtres 3 × 3 usuels (SOBEL, PREWITT, LAPLACE...).

— Extraction de textures : statistiques locales, masques de LAW, filtres passe-bande.

— Extraction de contours au moyen d'extracteurs performants (jusqu'à 11 × 11).

— Amincissement de contours par recherche du maximum local des gradients.

— Accumulation d'images, projections.

— Toutes les opérations arithmétiques et logiques entre deux images.

— Certaines combinaisons possibles des divers algorithmes ci-dessus.

Plusieurs INP20 peuvent être cascadés afin d'augmenter le nombre et la complexité des traitements ou de réduire le nombre de passes successives nécessaires à l'accomplissement d'une suite de traitements.

Le circuit a été réalisé au moyen d'une technologie « standard cells » CMOS 2 µm (E. BEAM). Grâce à son architecture élaborée et à la puissance de la décomposition par cascades de cellules, l'INP20 est un circuit de relativement petite taille par rapport à ses performances. D'une complexité équivalent à ≈ 12 000 portes, il occupe une surface de 85 mm^2. Il est encapsulé dans un boîtier « 84 pin LCC ».

Les dispositifs originaux concernant la réalisation des cellules de l'INP20 ont fait l'objet d'une demande de brevet. L'INP20 est disponible depuis le mois de juin 1988.

Une nouvelle version du processeur est actuellement en cours d'étude. Réalisé au moyen d'une technologie plus performante (1,5 ou 1,2 µm), elle sera capable de supporter des traitements non séparables utilisés notamment en morphologie (algorithme « rolling ball », fonction distance...).

8. Conclusion

Il existe une nette distinction entre les langages de description matérielle et les langages de description fonctionnelle. Les premiers décrivent l'assemblage de cellules physiques plus ou moins élaborées correspondant à une structure matérielle. Les descriptions obtenues sont utilisées pour le dessin des schémas d'implantation ainsi que pour la simulation logique ou électrique. Les seconds décrivent des comportements logiques. Ils correspondent généralement à des langages de programmation, permettant par simple compilation, de générer un code simulant le comportement recherché.

Le langage occam, bien qu'appartenant à la seconde catégorie, peut s'utiliser comme langage de description matérielle. Ceci résulte des notions de concurrence et de communication qu'il supporte, lesquelles reflètent le comportement des structures matérielles. Nous disposons donc avec occam d'un langage capable de décrire à la fois une structure matérielle et son comportement. Nous avons vu au travers d'un exemple comment il était possible d'exploiter cette propriété.

D'autres langages évolués sont susceptibles d'être exploités à cet égard, notamment le C, le Pascal et le Modula 2. Leur nature séquentielle occasionne un certain nombre de difficultés lors de la formalisation de systèmes dont les composantes fonctionnent essentiellement de manière concurrente. Elle impose une organisation spéciale des programmes et un choix précis parmi les structures de données disponibles. Elle aboutit à une description moins proche de la structure matérielle qu'une description fondée sur le langage occam.

Une approche quelque peu différente est proposée par les simulateurs au niveau « transfert de registres », tels Cassandre, Hilo... Elle consiste à animer le comportement d'un système décrit au moyen d'un langage de description matérielle. Le travail direct sur cette forme de description permet un recours immédiat aux outils de synthèse pour circuits intégrés, une fois le comportement jugé satisfaisant. La description matérielle provient généralement d'une représentation graphique acquise dans un environnement de conception assistée par ordinateur. Elle requiert déjà une assez bonne connaissance de la structure à implanter. Dans ce sens, elle est complémentaire de l'approche recourant à la programmation en langage évolué.

Il faut enfin noter que le problème du passage direct d'un comportement à une structure n'est pas résolu. Les approches décrites ne suppriment en rien l'étude architecturale préliminaire à chaque implantation. Toutefois, une formalisation utilisant le langage occam offre un moyen de se prémunir contre les erreurs dans une phase difficilement automatisable de la conception.

Manuscrit reçu le 20 juin 1989.

BIBLIOGRAPHIE

[1] A. PIRSON, J.-L. JACQUOT, T. COURT, D. DAVID, J.-L. JACQUOT, *A highly efficient method for synthesizing some digital filters,* conférence EUSIPCO septembre 1988.

[2] J.-L. JACQUOT, T. COURT, D. DAVID, A. PIRSON, *Une classe d'extracteurs de contours performants adaptés à une implantation matérielle fonctionnant à cadence vidéo,* C.E.A. D. LETI/SEIST, TIPI 1988.

[3] D. DAVID, T. COURT, J.-L. JACQUOT, A. PIRSON, INP20 : *An image neighborhood processor for large kernals,* IAPR workshop on CV. Special Hardware and Industrial Applications. October 12-14th, 1988. Tokyo, Japan.

[4] W. M. WELLS, Efficient Synthesis of Gaussian Filter by Cascaded Uniform Filters, in *IEEE PAMI* Vol. 8, n° 2, march 1986, pp. 235-239.

[5] L. A. FERRARI, P. V. SANKA, J. SKLANSKY, S. LEEMAN, *Efficient Two-Dimensional Filter Using B-Spline Function Computer vision,* in graphics and image processing, Vol. 35, 1986, pp. 152-169.

[6] D. J. BURR, *A fast filtering operator for robot stereo vision, Seventh International Conference on Pattern Recognition,* Montreal 1984, pp. 669-672.

[7] INMOS Limited, occam 2 Reference Manual, Prentice Hall.

[8] D. POUNTAIN, D. MAY, *A Tutorial Introduction to occam Programming,* BSP Professional Books.

[9] D. MAY, C. KEANE,, *Compiling occam into silicon,* Technical note 23, INMOS Limited, Bristol 1987.