Des architectures intégrées pour la vision : synthèse automatique en trois exemples

Integrated Architectures for Computer Vision : Automatic Synthesis With Three Examples

par François VERDIER* et Bertrand ZAVIDOVIQUE**

* Groupe de Recherche en Systèmes Complexes
Université de Cergy-Pontoise - IUT de Cergy
F-95014 Cergy-Pontoise Cédex
** Institut d'Électronique Fondamentale
Université de Paris-XI, Centre d'Orsay
F-91405 Orsay Cédex
Travaux réalisés au Laboratoire Système de Perception
DGA/CREA/ETCA
16 bis, Avenue Prieur de la Côte d'Or
F-94114 Arcueil Cédex

résumé et mots clés

La construction automatique d'ordinateur assistée par ordinateur $C(AO)^2$ est un problème ouvert parce que ceux-ci deviennent de plus en plus puissants, donc plus complexes et... plus petits. Nous expliquons ce qu'est la synthèse automatique de circuits intégrés dite de « haut niveau », technique désormais plausible et nécessaire notamment pour les architectures spécialisées en vision par ordinateur. S'agissant d'une optimisation dans un ensemble difficile à formaliser et à circonscrire nous décrivons la démarche expérimentale suivie afin de : 1) prouver l'existence d'une solution par cas d'application, 2) déterminer les paramètres de l'optimisation, dont l'état initial, et les instancier, 3) concevoir effectivement un circuit et 4) retraiter les solutions pour des architectures progressivement plus complexes n'en respectant pas moins des contraintes de temps réel. La démarche s'illustre par elle-même selon trois exemples de difficulté croissante qui jalonnent cet article.

Architecture, Synthèse VLSI, Traitement d'images, Graphes flot de données, Description fonctionnelle, Recuit simulé.

abstract and key words

Computer aided computer design is an open problem because computers are becoming more and more powerfull, more and more complex and.. smaller. We explain what "automatic (high-level) synthesis of integrated circuits" means. It is now feasible and necessary for computer vision dedicated architectures in particular. Since it requires an optimization within an ill-formalized and ill-defined design space, we describe the experimental method aiming at : 1) proving the existence of a solution for each application case, 2) finding and instanciating the optimization parameters – including the initial state –,3) effectively designing an integrated circuit and 4) redesigning the solutions for more complex architectures to still meet real-time constraints. The method is self-illustrated with three increasingly complex examples all along this paper.

Architecture, VLSI High-Level Synthesis, Image Processing, Data-Flow Graphs, Functional Description, Simulated Annealing.

1. introduction

Une voie de recherche fondamentale en architecture des ordinateurs, explorée depuis plusieurs années, est l'étude de la conception automatique de circuits intégrés; elle est motivée notamment par la complexité démesurée des circuits que l'on doit concevoir qui sont de plus en plus difficiles à réaliser « manuellement ».

Quant à l'intérêt d'architectures spécialisées il est à la fois scientifique et pratique, devant la puissance de calcul des processeurs banalisés qui autorise des traitements de plus en plus complexes. Le problème en est régulièrement posé lors de conférences d'ar-

chitecture comme *Computer Architectures for Machine Perception*.

En premier lieu, la conception d'architectures spécifiques traduit un rapport savoir-faire/technologie optimal : les connaissances techniques (informatique, électronique) et applicatives (le domaine jusqu'à son économie propre) contribuent aux circuits de moindre surface ou interconnexion (encombrement), de moindre consommation (puissance dissipée), de meilleure généralité (commande)... et font progresser la connaissance elle-même dans le domaine. C'est ainsi que, quelle que soit l'échelle d'intégration et d'implantation, on a effectivement constaté expérimentalement que l'efficacité des solutions architecturales est radicalement remise en cause par des contraintes liées à l'autonomie du système englobant.

D'autre part, une question robotique fondamentale reste de savoir s'il doit exister des opérateurs spécifiques de vision, puis si on saura les construire. Tout laisse penser, en l'absence de théorie des robots, qu'il n'y a d'autre démarche qu'expérimentale : fabriquer des circuits pour savoir quels circuits fabriquer.

Enfin si des facteurs économiques prépondérants peuvent inciter à programmer et interconnecter des microprocesseurs, il ne faut pas oublier que l'avancée technologique ne profite pas mieux aux grands constructeurs qu'aux laboratoires : l'avènement des ASIC, notamment, rend l'expérimentation accessible à tous, sauf peut-être pour ce qui est des logiciels de conception dont nous discutons certaines facettes dans cet article.

L'objectif de la synthèse automatique d'architectures est de traduire un ensemble de procédures ou un langage en un ensemble de circuits intégrés par une suite de transformations qui optimisent, sous contraintes, la représentation (réalisation) initiale mal adaptée.

La demande apparaît inévitable à nouveau pour rendre des systèmes autonomes, car l'autonomie suppose la perception des robots, dont une grande partie repose sur le traitement d'images. Or ce domaine est exemplaire de la complexité par le flot de ses données, dont le débit est couramment de 10^8 octets par secondes, mais encore par la sémantique de ses traitements. Révéler cette sémantique nécessite de nombreux changements de représentation, couramment du tableau de pixels au graphe des structures d'une scène. En revanche le traitement d'images présente sur le plan algorithmique l'avantage d'un développement de savoir-faire qui a conduit toute une communauté à construire, de manière parfois empirique mais toujours fonctionnelle, des enchaînements explicites d'opérateurs, progressivement concentrés dans des « boîtes à outils » (OTI [48], KHOROS [33]). On a donc là un corps de discipline qui, sans être trop restreint, peut favoriser la réalisation automatique de circuits intégrés, par rapport à d'autres domaines plus universels en apparence comme le « calcul scientifique » ou la « gestion de données ».

Si la synthèse de circuits a naturellement commencé par l'implantation de fonctions booléennes (avec les premiers compilateurs de silicium comme [22] et [40]), l'évolution de complexité des fonctions à intégrer fait qu'aujourd'hui on se préoccupe couramment de réaliser, à l'aide de procédés (quasi-)automatiques, des processeurs de calcul ou de traitement de signal, de type RISC par exemple, demandant plusieurs centaines de milliers de transistors.

Depuis les premières tentatives d'automatisation de la conception des circuits intégrés, de la saisie des masques de transistors (MAGIC [27] est l'un des plus populaires outils de ce type) aux premiers outils de compilation de silicium et de synthèse logique, les projets de synthèse d'architecture peuvent se classer historiquement dans trois catégories principales :

- Les développements d'algorithmes d'optimisation et de synthèse (Elf [18] qui utilise des règles grammaticales de transformation des architectures et HAL [30] qui incorpore les notions d'interconnexions à un algorithme d'ordonnancement par forces)
- Les outils de synthèse qui ont choisi de restreindre l'espace des solutions architecturales pour simplifier l'étape de construction (CATHEDRAL [19] synthétise des architectures mono ou multi-processeurs à partir d'un faible nombre de « briques de base »)
- Les projets plus complets comme USC [20] qui intègrent plusieurs outils spécifiques à chaque type d'architecture à construire (pipeline, multi-processeurs, utilisation intensive de mémoire)

Aucun de ces systèmes n'incorpore à la synthèse les notions de contrôle explicite : HAL s'en préoccupe indirectement via les interconnexions. Tous sont, comme notre solution, à modèle architectural prédéterminé. USC autorise plusieurs options de mise en œuvre distinctes. Sur le plan applicatif, ces systèmes restent peu spécialisés (modèle micro-processeur ou DSP). L'extension CATHEDRAL-IV au traitement d'images est limitée, aujourd'hui, à des filtrages et des opérations élémentaires encore proches du signal. Dans tous les cas, aucune des approches ne garantit *a priori* la validité des architectures synthétisées (considérées en tant que « systèmes » complexes), une fois placées dans leur environnement de fonctionnement réel.

Pour ce qui nous concerne dans cet article, les opérateurs de vision s'entendent susceptibles d'extraire véritablement une information pertinente sur la forme, la dynamique ou l'apparence d'objets ou de situations : ils sont donc de taille équivalente aux circuits DSP courants. C'est pourquoi nous marquons une différence entre synthèse d'architecture, à peu près « High Level Synthesis » dans la littérature anglo-saxonne, et synthèse de circuits qui peut n'être que traduction logique ou compilation de silicium, mieux formalisable. **Nous décrivons dans la suite à la fois une démarche et son aboutissement sous la forme d'un environnement complet de développement et de synthèse d'architectures de vision produisant des circuits.**

Cette démarche est résolument pragmatique. Une présentation globale assez formelle des problèmes, suivie d'un exposé des solutions proposées, classiquement illustrés par une ou deux réalisations pourrait laisser penser que l'ensemble des problèmes en synthèse est définitivement connu, que les limites des hypothèses

sont cernées, que les choix techniques sont absolus ou définitifs, et que la méthode est complète et sûre. Nous choisissons donc de construire cet article comme une progression de trois circuits de complexité fonctionnelle croissante, qui ont été effectivement synthétisés. Au-delà d'une simple présentation de résultats de recherche, cet aspect « informatique expérimentale » nous paraît devoir être mis en exergue. Au fil des trois exemples il ressort à la fois par la difficulté croissante des problèmes rencontrés et par leur illustration immédiate sur des cas concrets. Le plan de l'article est donc le suivant :

- Le chapitre 2 pose l'énoncé du problème de la synthèse, supporté par un premier exemple élémentaire d'un circuit réalisant une fonction de **codage** (un étiqueteur en composantes connexes). On explicite, grâce à l'exemple, la notion de complexité image, la relativité des choix de représentation (notamment fonctionnelle) et les limites *a priori* des problèmes identifiés à ce stade : ces problèmes dans les cas suffisamment simples sont plus relatifs aux *circuits* qu'aux *architectures*.
- Le chapitre 3 expose la méthodologie prônée, ainsi que ses premiers résultats chiffrés, appliquée à un opérateur de **traitement** plus complet mais encore très fruste (un détecteur de contours). On explicite sur cet exemple l'ensemble des différents facteurs d'optimisation, d'ailleurs non exhaustif, et les limites en résultant.
- Le chapitre 4, enfin, propose certains résultats prospectifs permettant de mesurer l'impact de contraintes additionnelles plus liées au réalisme d'un système complet. Parmi de nombreuses améliorations possibles sont choisies la hiérarchisation des traitements (par duplication de code) et l'accélération des exécutions (par une mise en pipeline des architectures). Elles sont décrites à partir d'un opérateur de **vision** réflexe plus proche d'un composant pour robots (un détecteur de déformation).

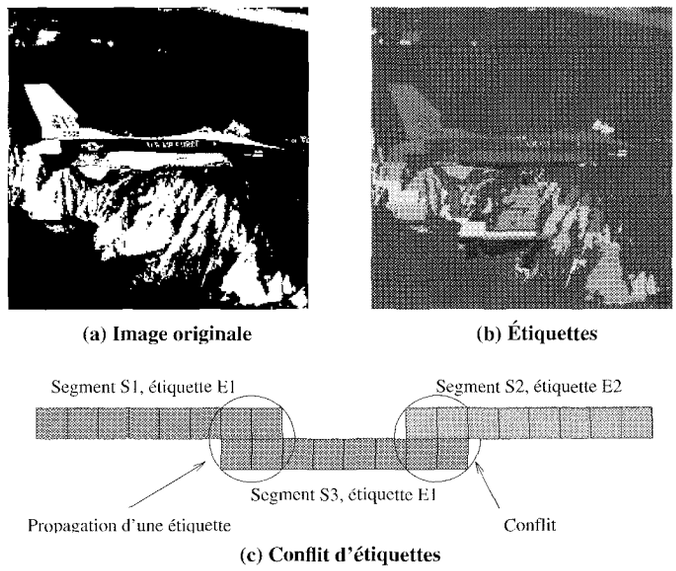
Ces trois exemples ne valent pas preuve, même au sein d'une démarche expérimentale, mais ils préviennent contre les formalisations ou généralisations prématurées. Leur progression devrait quant à elle suggérer l'intérêt du problème et mieux recadrer les résultats actuels. En revanche, le fait qu'un seul logiciel ait synthétisé ces trois circuits, parmi d'autres, constitue sûrement une présomption de validité de l'approche.

2. le problème de la synthèse : exemple d'un étiqueteur

Supposons qu'une première décision ait été prise sur une image, résultant en une représentation binaire (pixel de valeur 1 : point intéressant ou objet, valeur 0 : point négligeable ou fond). De cette nouvelle image un système doit déduire des plages ou régions qui sont les projections des objets dans le plan focal : cette opération, sorte de coloriage quasi instinctif dans la sphère animale, s'appelle l'étiquetage en composantes connexes et se « programme » volontiers sous la forme d'automates séquentiels exécutant l'opération à la volée. L'algorithme correspondant servira ici pour expliquer concrètement le problème de la synthèse d'architectures.

2.1. présentation de l'algorithme

Le fonctionnement est illustré figure 1. (cf. [32]).



(a) Image originale (b) Étiquettes

(c) Conflit d'étiquettes

Figure 1. – Application de la première passe d'étiquetage sur une image binaire.

Le principe en est le suivant :

- sur l'image binaire originale, les segments horizontaux (suites de pixels « blancs » ou de valeur 1) sont extraits,
- lors du balayage d'un segment, si une connexion avec un segment précédemment étiqueté est détectée alors l'étiquette (la couleur) est propagée verticalement, sinon une nouvelle étiquette est générée,
- sur l'image des étiquettes (une étiquette par segment) on applique une phase de « recalage » et de propagation horizontale des étiquettes sur toute la longueur du segment,
- pendant l'inter-trame (délai entre deux images consécutives d'une séquence) les équivalences résultant des conflits (figure 1.c) entre les étiquettes sont résolues.

Le principe d'étiquetage par segments complets réduit sensiblement le nombre d'équivalences à résoudre et permet une exécution en temps réel de l'algorithme (en particulier sur la machine d'émulation présentée au § 3.2).

2.2. les problèmes posés par la synthèse

Préalable à toute construction, tant manuelle qu'automatique, d'une architecture qui implante cet algorithme, une analyse est nécessaire (illustrée figure 2) concernant essentiellement :

1. la spécification initiale de l'algorithme, et comment s'en accomoder si elle n'est pas parfaitement adéquate,
2. la représentation intermédiaire du problème (la structure de données interne) la plus efficace pour la synthèse et ses optimisations successives,
3. la méthode de conception, et les différentes techniques d'optimisation, pour résoudre le problème,
4. les critères de qualité d'une architecture pour évaluer les résultats au regard des besoins,
5. la structure de l'architecture à construire (son modèle d'exécution, de contrôle) et la latitude de choix en cette matière.

Spécification comportementale.

Le premier problème à résoudre est celui du choix de la description (spécification) comportementale initiale. L'algorithme que l'on se propose de synthétiser doit être décrit sous une forme qui permette de le manipuler aisément. La description de l'algorithme d'étiquetage que nous avons donnée au paragraphe 2.1. est du type « conceptuel » et s'intéresse essentiellement à la sémantique du traitement et à l'enchaînement des tâches. Malgré sa puissance d'expression, il est clair qu'une spécification « en français » est pour longtemps encore difficile à manipuler par un logiciel et que l'étape de synthèse serait particulièrement difficile. A l'inverse, la spécification de départ pourrait être une ébauche de schéma comme illustré par la « spécification schématique » de la figure 2. Dans ce cas, on s'intéresse plus particulièrement à la structure qui permet d'implanter le traitement qu'à l'algorithme lui-même : on manque donc d'informations pour faire évoluer l'architecture. De plus, cette ébauche peut s'avérer une condition initiale défavorable : les choix structurels déjà opérés ne sont pas forcément adaptés aux contraintes que doit respecter l'architecture finale (tout traiteur d'image n'est pas bon architecte pour esquisser cette spécification!).

Entre ces deux extrêmes, on dispose de toute une variété de langages de programmation qui permettent de spécifier un traitement (l'algorithme) mais qui supportent plus ou moins efficacement une synthèse automatique (voir l'encadré « Spécification algorithmique » de la figure 2). Le langage C a été largement utilisé comme support de représentation des procédures en vue de leur intégration automatique. Il est parfaitement maîtrisé par la plupart des traiteurs d'images et sa puissance en fait un outil bien adapté au codage de tels algorithmes. Malgré ces avantages, sa structure fondamentalement impérative limite son utilisation en synthèse automatique : la présence de variables, et surtout d'assignations, masque le parallélisme intrinsèque des algorithmes (l'enchaînement des opérations n'est identifiable que grâce aux variables). On en trouve d'ailleurs une illustration dans le programme C de la figure 2 avec la présence – et l'utilisation – de la variable `conv` qui décompose la procédure en deux entités séquentielles (la boucle `while` et la structure `if`) qui sont néanmoins exécutables à la volée. Le langage Ada a été, lui aussi, largement employé en synthèse d'architecture [18]. Tout comme VHDL [25], ses structures fortement procédurales limitent le parallélisme des applications et l'efficacité des architectures conçues. Cette analyse (on pourra trouver dans [1] les premiers travaux sur l'expression fonctionnelle des traitements puis dans [39] des développements plus récents), conduit à utiliser un langage qui soit à la fois adapté à la représentation des algorithmes de traitement d'images et indépendant de toute structure capable de l'exécuter (pour ne pas restreindre la recherche d'une architecture optimale). Celui que nous pronons, FP, est un langage fonctionnel proche de celui défendu par B dans [5]. Nous développerons ce choix et les avantages qu'il présente pour l'aspect de prototypage rapide au paragraphe 3.4. D'autres choix sont possibles mettant l'accent sur d'autres aspects, comme la gestion du temps et des synchronisations (par exemple avec SIGNAL [7])ou comme l'intuition graphique ou le confort (CANTATA [33] et [12, 13]).

Représentation du problème.

Cette spécification de l'algorithme n'est en réalité que la « valeur » initiale d'une structure de représentation interne apte à supporter toutes les étapes ultérieures de conception et d'optimisation. Il est commun d'utiliser pour de telles structures le graphe de flot de données (DFG) qui décrit les dépendances de données entre les opérations, mais il s'est avéré judicieux de généraliser ce formalisme. En effet, nous sommes intéressés par l'optimalité des architectures à concevoir en terme **non seulement de qualité du chemin de données mais également de qualité du contrôle**. Nous verrons dans la suite de cet article que la représentation mixte de ces deux aspects d'une architecture a été rendue possible grâce aux graphes dits CDFG (§ 3.5) et que, de plus, une nouvelle méthode d'optimisation du contrôle a pu être ainsi mise au point par la maximisation de la régularité de tels graphes (§ 3.7).

Méthode d'optimisation et critères.

En ce qui concerne l'étape de construction de l'architecture par elle-même, il faut fixer un niveau d'abstraction (synthèse système, architecturale, RTL[1], logique?) et une méthode de conception adaptée à ce niveau. La conception peut se faire de manière purement descendante (hiérarchique), montante (assemblage de blocs) ou mixte. On doit également adopter une méthode d'optimisation de l'architecture (améliorations itératives, programmation linéaire, *branch and bound*,...?). De plus, toute optimisation mobilise un certain nombre de critères, ses variables, qui doivent être correctement et judicieusement choisis : en pratique surface, performance, nombre d'entrées/sorties détaillés au § 3.6.

1. RTL : *Register Transfer Level* pour niveau des transferts entre registres.

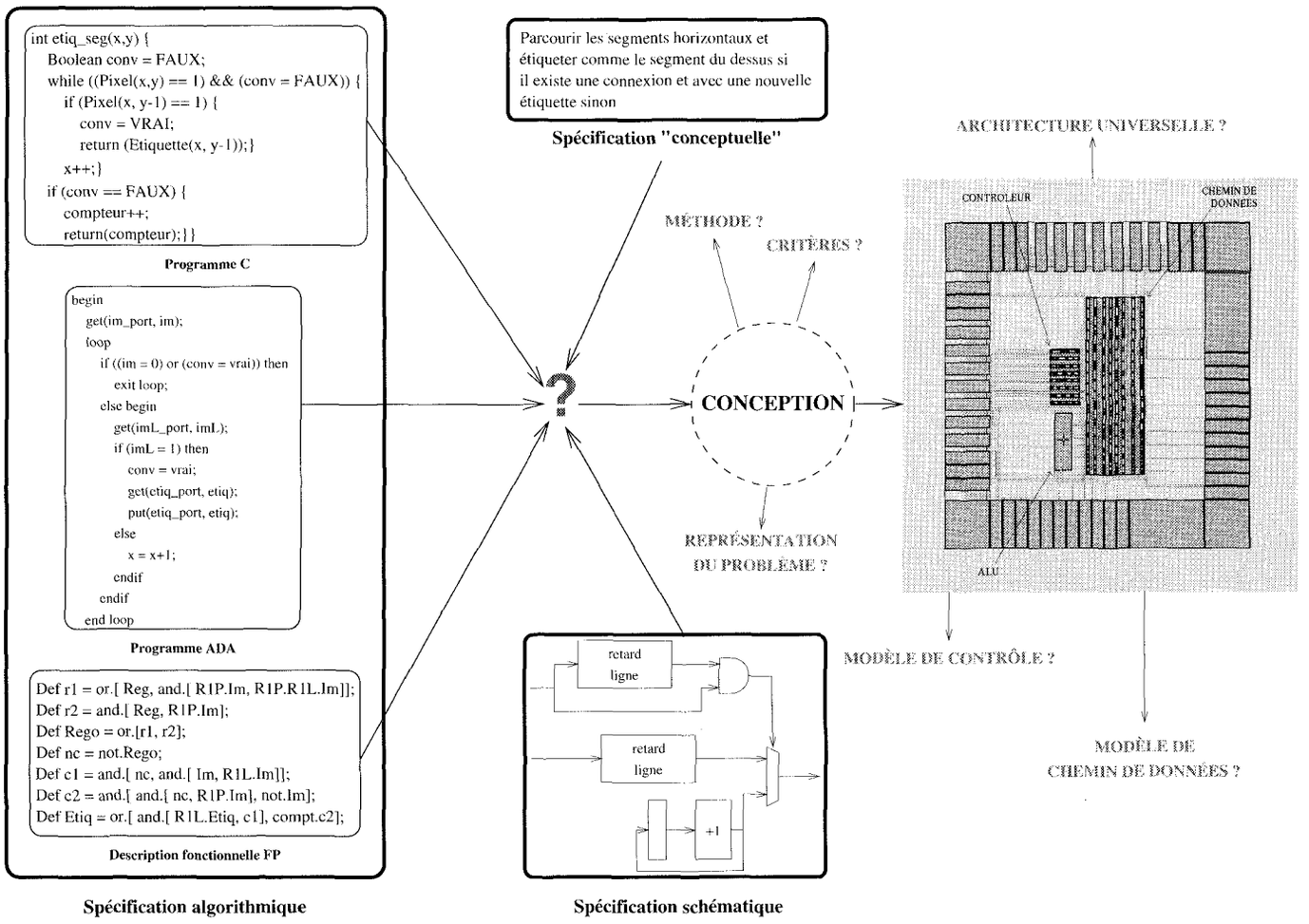


Figure 2. – La conception automatique d'un étiqueteur en composantes connexes et ses problèmes.

Modèle structurel des architectures.

En ce qui concerne l'architecture elle-même, le choix d'un modèle d'exécution ne doit pas être laissé au hasard. Le modèle de contrôle doit être précisé (contrôle localisé, réparti, par les données, ...) pour restreindre à bon escient l'espace de conception. Nous adoptons à titre expérimental pour architecture cible les **chemins de données contrôlés par machines à états**. C'est un choix logique si l'on doit optimiser le contrôle pendant la conception. Ce modèle permet effectivement de résumer en une seule structure de données les informations de calcul combinatoire et de séquencement. Les machines de ce type supportent mieux les optimisations de séquencement que d'autres comme les machines à piles ou à contrôle micro-programmé. Ce choix n'est d'ailleurs pas limitatif comme en témoigne l'extension pipeline du § 3.2 donc pas exclusif ni définitif. Il a, au stade actuel des développements, l'avantage de ne pas préjuger du type de parallélisme (données, contrôle...) et de ne pas contrarier d'emblée la spécification comportementale en privilégiant strictement le respect des dépendances de données lors de l'évolution du circuit (cf. § 3.2.).

Tout comme la spécification « initiale » d'ordre algorithmique, la spécification « finale » d'ordre architectural est une contrainte qui doit limiter l'espace de conception (régulariser), mais pas trop! Une procédure en C, par exemple, est la projection d'un algorithme sur un ensemble d'opérateurs (très) virtuels, dont certains ne doivent rien à l'opération mais concernent la machine (Von Neumann), le compilateur voire même le système d'exploitation (cf. exemple dans le paragraphe « spécification comportementale »).

En « projetant » d'abord sur le langage FP (il a fallu repenser l'algorithme d'étiquetage pour le coder) puis sur le modèle d'automates (par optimisation), on essaie en réalité de rester proche d'une architecture optimale pour moins risquer de tomber dans un minimum local (i.e. ne satisfaisant pas au mieux tous les critères explicités). L'existence même de cet optimum résulte de la démarche (cf. 3.2). En l'absence de théorie plus formelle, tout est question de subtil dosage intuitif pour : rester général en Traitement d'Images, rester général en architecture, et... réussir.

2.3. adéquation algorithmes/ architectures

En réalité, le problème ici posé est celui de l'adéquation entre algorithmes et architectures. Plus universelle, la question est : existe-t-il *une* architecture *a priori* adaptée à la vision des machines, sorte de microprocesseur qu'il suffirait de « retailler » sur mesure (voire même de paramétrer) pour une phase ou une opération de vision donnée?

Si une telle architecture existe nous n'avons pas été capables – aujourd'hui – de la trouver[2]. Pour le traitement numérique du signal, on peut penser qu'une architecture *à tout faire* existe : celle des processeurs dits DSP. En réalité, ils implantent essentiellement le produit scalaire, ce qui précise en même temps leur puissance mais aussi leurs limites : pensons à tout ce qui n'est pas filtrage linéaire comme la programmation dynamique en parole ou les approches bayesiennes. Le cas est encore différent pour le Traitement d'Images et les raisons de cette lacune sont multiples :

1. En premier lieu l'énorme quantité de transformations mises en jeu lors de la manipulation d'une image. A titre d'exemple, une élémentaire fonction de vision statique (comme celle du chapitre 3) met en jeu une dizaine d'opérateurs fonctionnellement comparables à des convolutions (filtrage directionnel, extraction de maximum, chaînage, comparaisons pour analyse de formes). Elle demande donc une puissance au moins égale au GOPS. En effet, une simple convolution 3×3 (un filtrage tel que celui qui fabrique l'image binaire de départ de l'étiquetage) nécessite une puissance de calcul supérieure à 100 MIPS pour une exécution en temps réel (25 Hz) sur des images 512×512! L'ordre de grandeur du débit des données pixel est de 6,5 Mpixels/s! Dès que l'analyse devient dynamique (mouvements dans une séquence d'images, mais encore couleur en imagerie multispectrale, volumes par stéréovision) la complexité en nombre d'opérations gagne encore un ordre de grandeur. Les architectures à concevoir doivent donc posséder un degré de parallélisme majeur (de quelques dizaines d'opérateurs distribués autour d'un média (réseau ou mémoire) à plusieurs milliers dans les systèmes cellulaires), limité par la technologie.
2. Ensuite la grande diversité de structures de données évolutives traitées par les algorithmes. Dans le cas de notre étiqueteur, la structure de départ est un tableau bidimensionnel de pixels (l'image) et évolue au cours de l'algorithme vers une structure de lignes (à cause du balayage) puis de vecteurs (les segments de droites) puis enfin vers une structure de surfaces (vecteurs de paramètres ou graphes de descriptions) telle que la traiterait un algorithme de reconnaissance de formes par exemple. Les architectures très fortement contraintes quant à leurs données telles que les machines systoliques ou vectorielles (qui peuvent toutefois exécuter – même si c'est peut-être sous-optimal – un étiquetage) sont donc à écarter.

   On envisagera plutôt des architectures dites *à contrôle à la demande* (on trouvera de nombreuses références dans [36]).
3. Les nombreuses contraintes que l'on désire respecter et, en particulier, celles de l'exécution en temps réel et de l'embarquabilité induisent naturellement une échelle d'intégration des architectures à concevoir dont la disponibilité est au mieux des plus récentes.

   La nécessité de solutions VLSI (*Very Large Scale Integration*) ou même WSI (*Wafer Scale Integration*) s'en trouve confirmée.
4. Enfin, il n'existe pas de méthode universelle pour le développement des algorithmes de traitement d'images (notamment par manque de formalisme) adaptés à un problème particulier et donc *a fortiori* de méthode de conception d'une architecture adaptée à un besoin!

   La conception des architectures fera intervenir un grand nombre d'heuristiques pour déterminer **le** circuit qui implantera **un** algorithme particulier qui résoud **un** problème donné.

En conclusion, on a restreint en premier lieu le domaine algorithmique à celui des traitements de bas et moyen niveau, puis a été développé un outil de synthèse automatique d'« architectures sur mesure[3] » pour un algorithme donné (on parlera alors plutôt d'**automate de traitement**). Les traitements de haut niveau relèvent, quant à eux, plus de la décision, couvrant toutes les formes d'optimisation de l'analyse statistique aux techniques symboliques en passant par l'analyse grammaticale. Nous les implantons donc dans l'automate de traitement sur des processeurs généraux, nécessaires par ailleurs au contrôle du système de vision. Les architectures correspondantes sont en effet encore plus difficiles à concevoir automatiquement, du moins par notre méthode, ne serait-ce que par l'inexistence *a priori* de potentielles expressions fonctionnelles.

3. première solution : exemple d'un détecteur de contours

La détection de contour sert ici pour expliquer concrètement la difficulté méthodologique qu'est la validation (quelques réfé-

2. Dans le cas contraire, nous n'aurions pas utilisé dans cet article trois exemples différents de circuits synthétisés.

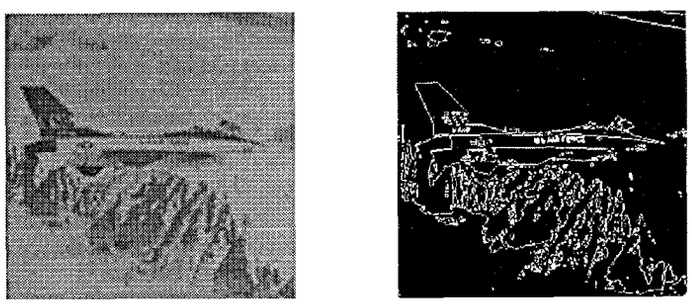
3. Il s'agit ici plutôt de *prêt à intégrer*. Notons qu'il existe dans le secteur purement technologique de la conception, les cellules pré-caractérisées par le fondeur (au niveau porte logique). Nous aurions donc rendu le concept plausible au niveau RTL, au prix d'une éventuelle restriction du domaine applicatif.

rences suffisent en pratique pour un tel traitement d'image des plus classiques), et surtout pour poser la structure de données clef (le CDFG), la spécification fonctionnelle préalable, et les transformations lors de l'optimisation, qui resteraient sans cela inutilement abstraites.

Est-ce que, si l'on a correctement choisi le niveau d'expression de l'algorithme, une bonne représentation interne et une architecture cadre efficace, on est garant, pour autant, du succès du processus de synthèse? Si l'opération à implanter est suffisamment élémentaire et fonctionnellement sûre sans plus de validation (l'exemple précédent est une partie seulement d'un algorithme plus complet) c'est probable. Ce n'est déjà plus vrai lorsque l'on monte en complexité comme avec l'opérateur suivant de détection de contours. Il réclame une présomption de validité système : en effet, cet opérateur n'est pas forcément le meilleur dans le cas précis de mise en œuvre et des centaines de circuits concurrents pourraient se concevoir! Nous proposons une solution fondée sur l'émulation pour surmonter cette difficulté méthodologique. On envisage ensuite l'intégration : ceci est le rôle du logiciel ALPHA.

3.1. présentation de l'algorithme

Un exemple de résultat apparaît figure 3. La détection d'un contour (frontière d'une région) dans l'image est ici basée sur un calcul de gradients (dans les quatre directions d'un voisinage de cinq pixels) puis achevée par seuillage du maximum de ces quatre contributions. Les gradients (dérivées spatiales) sont calculés par la valeur absolue d'une différence entre pixels. Le voisinage est implanté, quant à lui, par des opérateurs spécifiques de retard (retard de un pixel pour les gradients horizontaux et retard de une ligne pour les gradients verticaux).



(a) Image originale (b) Image des contours

Figure 3. – Application de la détection des contours sur une image test.

Au plan applicatif, il faut donc d'abord savoir si malgré sa simplicité (aucune vocation d'optimalité [11] [14], aucune précaution d'immunité au bruit [17] [26]) ce détecteur sera satisfaisant sur les scènes envisagées, s'il respectera les délais d'exécution fixés ou offre une bonne présomption de le faire dans sa version intégrée. Il faut ensuite le faire évoluer sous contraintes.

3.2. méthodologie globale

La méthode générale que nous choisissons pour concevoir ces automates de traitement est celle de la conception à partir des résultats d'émulation (illustrée figure 4).

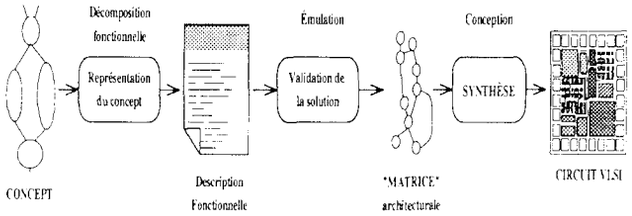


Figure 4. – Principe global de synthèse à partir de résultats d'émulation.

Ce principe se décompose de la manière suivante :

- à partir d'un problème de traitement d'images donné (effectuer la détection des contours d'une image) et en utilisant la méthode de décomposition fonctionnelle [37, 47, 28, 29], une première ébauche d'un algorithme est codée sous forme fonctionnelle. La sémantique du traitement, sa hiérarchie éventuelle et la bibliothèque des primitives fonctionnelles disponibles sont abondamment utilisées pour le codage.
- cette représentation fonctionnelle en langage FP (sa traduction sous forme de graphe de flot de données plus précisément) est ensuite utilisée pour programmer une machine dédiée à l'exécution de l'algorithme en temps réel. Cette machine (le Calculateur Fonctionnel présenté figure 5) est dotée d'un réseau tridimensionnel de 1024 processeurs de bas niveau à contrôle flot de données couplé à une maille de processeurs Transputers [39]. L'exécution réalise en pratique l'émulation d'une architecture implantant l'algorithme. Il y a trois principaux avantages à cette émulation :
  1. La preuve de l'existence d'une instance d'architecture capable d'implanter l'algorithme,
  2. La preuve de validité de l'algorithme et, plus important, la preuve de l'adéquation de la solution algorithmique avec le problème précis à résoudre (tout en respectant la contrainte de temps réel),
  3. La mise au point « instantanée » de l'algorithme, rendue possible grâce à un environnement de programmation évolué et au principe d'exécution en ligne au rythme des capteurs. Les effets d'un paramétrage d'opérateur sont appréciables dès le passage des valeurs (en quelques millisecondes) sur le flot des images issues d'une caméra. Une modification plus profonde d'un opérateur nécessite, elle, la recompilation et l'installation sur le réseau [38] (quelques minutes).
- le résultat de l'émulation constitue un cadre architectural – la « matrice » – à partir duquel on effectue enfin une synthèse automatique dont l'objectif est principalement de construire une version plus compacte de ce qui a été effectivement

utilisé dans le Calculateur Fonctionnel. La traduction automatique d'un programme FP en un graphe de contrôle et de flot de données constitue la première étape de synthèse. Grâce à une bijection entre ces deux représentations, les avantages liés aux macro-fonctions, la compilation conditionnelle et l'expression hiérarchique des traitements sont conservés pour la synthèse. A ce stade, aucune intervention manuelle n'est nécessaire.

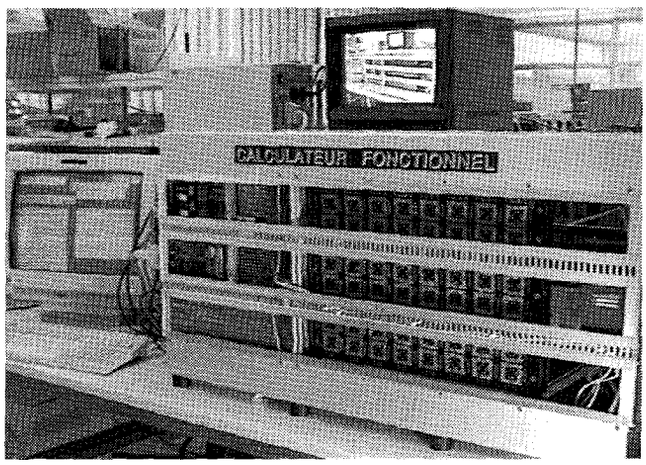


Figure 5. – L'émulateur Calculateur Fonctionnel vu par lui-même.

Le principal atout de cette démarche est d'assurer en amont de la synthèse une preuve de validité des algorithmes et, en aval, la correction des architectures vis-à-vis des contraintes initiales. A la condition de respecter la sémantique des graphes manipulés pendant la synthèse (c'est-à-dire *de ne pas briser les dépendances de données*), l'architecture construite est fonctionnellement *valide*. En conséquence, si on a su exhiber et programmer une procédure initiale sur le calculateur flot de données, on assure la validité par construction des circuits synthétisés (on a déja souligné qu'il avait fallu repenser l'algorithme d'étiquetage).

Il est intéressant de comparer cette démarche aux méthodes dites de prototypage « rapide » sur cibles FPGA [46] qui permettent de valider un système VLSI avant son intégration mais au prix d'un cycle de développement du prototype parfois long. Il est cependant difficile d'effectuer des comparaisons chiffrées entre notre méthode d'émulation/synthèse et les méthodes VHDL/FPGA faute de repères communs en termes d'algorithmes et de technologies. Reste que, d'après les auteurs de la machine SPLASH-2 eux-mêmes, les temps de simulation en VHDL, de partitionnement en plusieurs circuits FPGA et de conception/placement/routage des circuits peuvent devenir prohibitifs [4].

3.3. le logiciel ALPHA

L'organisation générale de l'outil de synthèse ALPHA (illustré figure 6) se décompose comme suit :

- une interface en amont avec l'environnement de programmation du Calculateur Fonctionnel, à savoir un compilateur de programmes fonctionnels (en langage FP que nous présentons au § 3.4) en graphes de flot de données (présentés au § 3.5).
- une interface en aval avec un outil industriel de CAO. On ne manipule pas en effet les informations de très bas niveau des architectures (portes logiques, placement et routage de cellules, dessin des transistors), on ne se concentre que sur la synthèse RTL puisque tel est le niveau où nous décidons d'intervenir avec ALPHA. La chaîne de CAO que nous utilisons dans cet environnement est l'outil de conception COMPASS. Nous donnons au § 3.6 le résultat de la « compilation » par COMPASS du détecteur de contours.
- le cœur de l'outil de synthèse qui est basé sur deux algorithmes d'optimisation stochastique par recuit simulé : l'ordonnancement qui détermine l'instant précis d'exécution de toutes les opérations d'un graphe, et l'assignation qui sélectionne un opérateur VLSI particulier pour implanter chacune de ces opérations. Nous présentons brièvement le recuit simulé au § 3.6, ces deux algorithmes ayant été largement détaillés dans [34] et [35].
- une bibliothèque de modules VLSI dans laquelle sont choisis les composants qui constitueront l'architecture finale (un soustracteur à seuil, un comparateur ou même un trieur sont des candidats à l'implantation de l'opération MAX). Cette bibliothèque peut être étendue avec les macro-cellules elles-mêmes synthétisées par ALPHA permettant ainsi la construction hiérarchique des circuits (comme nous le verrons au § 4.2 avec un circuit plus complexe qui demande ce genre de progression : un détecteur de défauts).

3.4. spécification fonctionnelle des algorithmes

La figure 7 illustre le code FP (cf. § 2.2) du détecteur de contours. Les premiers travaux relatifs à la programmation fonctionnelle en traitement d'images ont été publiés dans [1].

Les avantages d'une telle spécification sont les suivants :

- la spécification naturelle des traitements. On se représente (et on code) en effet naturellement une détection de contours par : -1) un ensemble de fonctions d'extraction d'un voisinage dans l'image (lignes 1 à 3 de la figure 7), -2) les fonctions de calcul des quatre dérivées spatiales (lignes 4 à 7), -3) la normalisation par un facteur de $\frac{1}{\sqrt{2}}$ des gradients diagonaux et le calcul du maximum des gradients (lignes 9 à 13) et enfin (ligne 15) le seuillage du résultat. Toutes ces fonctions sont pensées naturellement en séquence et codées de cette façon.

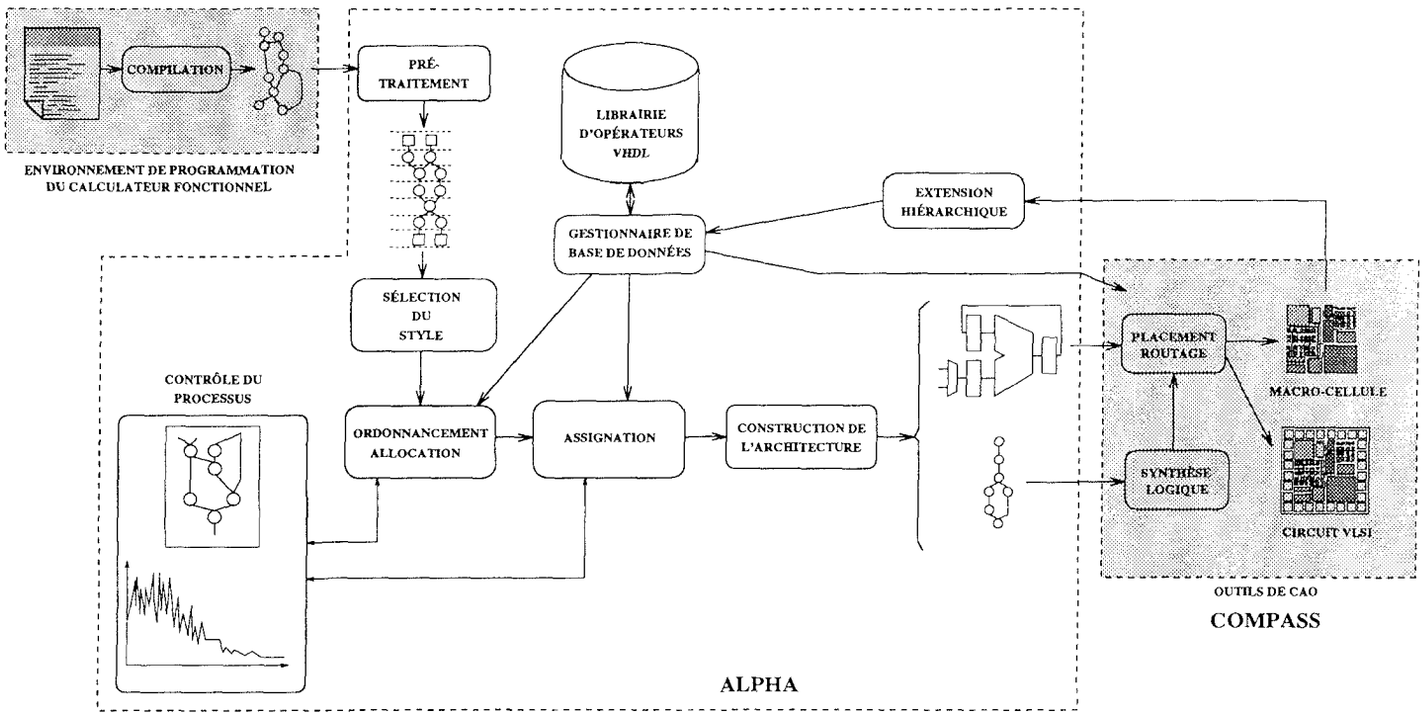


Figure 6. – Organisation générale du logiciel de synthèse ALPHA.

```
// EXTRACTION ÉLEMENTAIRE DE CONTOURS PAR
// SEUILLAGE DU MAXIMUM DE 4 DÉRIVÉES
main [
VIDEO INPUT X :PIXEL;

VIDEO INPUT T :PIXEL;

VIDEO OUTPUT Y :BIT;
]
 1- DEF XL = R1L . X;
 2- DEF XLR = R1P . XL;
 3- DEF XP = R1P . X;
 4- DEF D1 = abs . sub . [XP, R1P . XP];
 5- DEF D2 = abs . sub . [XP, XLR];
 6- DEF D3 = abs . sub . [XP, R1P . XLR];
 7- DEF D4 = abs . sub . [XP, XL];
 8-
 9- DEF MAX1 = max . [D1,D2];
10- DEF MAX2 = max . [D3,D4];
11- DEF mm = shr.shr.MAX2;
12- DEF mmm = add. [ add. [mm,mm], mm];
13- DEF MAX = max . [MAX1 ,mmm ];
14-
15- def Y = thr(1) . [ MAX, T];
```

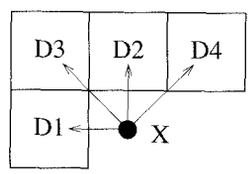


Figure 7. – Spécification en langage FP du détecteur de contours et masque du calcul des gradients.

– le caractère hiérarchique par nature. La représentation fonctionnelle d'un traitement est fortement liée à la disponibilité de certaines fonctions de base (les primitives). A un niveau de description fin, ces primitives sont des opérations arithmétiques ou logiques élémentaires (additions, soustractions, maximum et seuillage). Or, au début du processus de codage FP, le traiteur d'image peut se représenter la « fonction » de calcul du gradient maximum sur le voisinage par `MAX_GRAD (D1, D2, D3, D4)` sans entrer dans le détail du calcul et de la normalisation diagonale. Cette hiérarchie de spécification (à condition d'être supportée par la librairie des primitives) est un avantage significatif lors de la spécification « intuitive » d'un traitement.

– l'encapsulation fonctionnelle. L'émulateur Calculateur Fonctionnel (cf 3.2.) est doté d'un réseau de processeurs Transputer programmés en langage C. Ceux-ci exécutent les primitives difficiles (ou impossibles) à implanter sur un ou plusieurs processeurs de bas niveau. Ils sont également utilisés pour les traitement dits de haut niveau (mise en forme de résultats, traitements symboliques...) et dans les phases de décision (reconnaissance, contrôle visuel). De telles primitives C sont toutefois totalement compatibles – pour ce qui est de leur description FP – avec les primitives fonctionnelles de bas niveau (on peut se référer à [37] pour les problèmes de synchronisation en particulier). Dans l'architecture finale, elles ne sont pas re-synthétisées mais conservées sur des modules à base de Transputer.

3.5. les graphes de contrôle et de flot de données

La description FP, bien que naturelle et indépendante d'une architecture permettant de l'exécuter, n'est pas adaptée à la représentation d'une architecture en cours de synthèse. Dans la plupart des outils de synthèse classiques, la représentation interne est basée sur une structure de graphe de flot de données : YIF [9] pour le *Yorktown Silicon Compiler*, SSIM [10] pour le système HIS, « Value-Trace » [6] [41] du *System Architect's Workbench* ou la structure DDS [23] du système ADAM.

Toutes ces structures de données possèdent certains inconvénients :

1. Ces structures de représentation interne manquent de cohérence et d'uniformité dans la sémantique de leurs éléments constitutifs (les nœuds de contrôle sont souvent sans rapport avec les nœuds symbolisant des opérations flot de données). Dans la structure interne VT, plusieurs cas particuliers doivent être pris en compte : l'ordonnancement d'un nœud opération est différent de l'ordonnancement d'un nœud de branchement (nœuds RESTART et LEAVE pour le début et la fin d'une procédure) en regard de la structure de l'architecture (ces derniers n'ont pas de correspondance sur le chemin de données de l'architecture) mais également de son coût (surface, performance). L'algorithme d'optimisation devient complexe et difficilement testable.
2. La quantité d'informations à représenter augmente considérablement la taille de la structure interne et complique les étapes de « mise à jour » des objets. Choisir, par exemple, de ne pas considérer les étapes de synthèse logique peut réduire la taille d'une structure de données interne en supprimant les informations relatives à l'implantation technologique des architectures. A l'inverse, la structure DDS permet de représenter toutes les informations relatives à une architecture depuis la spécification comportementale de haut niveau jusqu'aux spécifications des portes et transistors qui composent l'architecture.
3. Enfin, à plus haut niveau, la spécification initiale – lorsqu'elle est donnée dans un langage inadapté – est difficile à écrire et surtout non intuitive. Un typage strict des données s'adapte mal au Traitement d'Images (matrices de pixels, liste de points, histogrammes, surfaces de régions, etc...). Il existe par exemple plusieurs façons de programmer en C le balayage d'une image ainsi que la représentation de l'image elle-même (tableau bidimensionnel, pile de stockage temporaire,...). Chacune de ces représentations aura une réalisation matérielle (si on réussit à la trouver) différente et de « qualité » différente. Les structures images utilisées dans le code C des Transputers par exemple sont de type pile FIFO (via les ports d'entrée/sortie pour l'accès en mode flot de données) alors que celui de l'étiqueteur (cf. figure 8) est un tableau bidimensionnel.

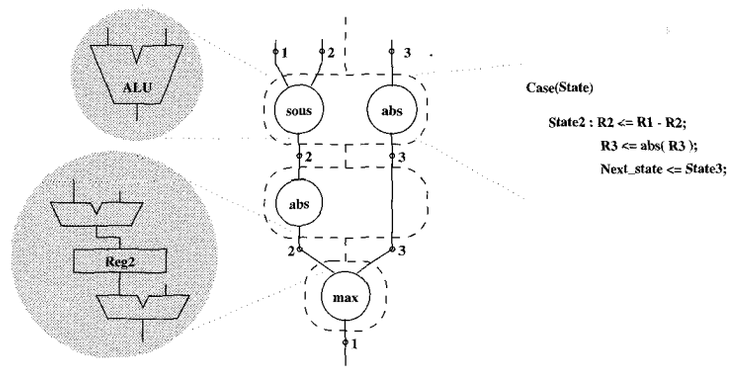


Figure 8. – Une partie du graphe de contrôle et de flot de données du détecteur de contours.

Dans le système ALPHA, l'architecture « en chantier » est supportée par un graphe représentant les dépendances de contrôle et de données (CDFG : *Control Data Flow Graph*). Les travaux relatifs à cette structure ont été déjà publiés dans [44]. Les objectifs en sont de réunir au sein d'un formalisme unique les éléments de représentation algorithmique (séquentielle, contrôle) et architecturale (structurelle, chemin de données) pour :

1. disposer de **toutes** les informations sur l'architecture à synthétiser,
2. être capable de « mesurer » **efficacement** la qualité d'une solution,
3. optimiser avec un seul algorithme le contrôle **et** le chemin de données.

La figure 8 schématise la partie du graphe de contrôle et de flot de données de l'algorithme de détection de contours réalisant la fin du calcul du gradient maximum sur le voisinage. Les nœuds en trait plein sont les opérations élémentaires du graphe de flot de données (les primitives de la desciption FP) reliées entre elles par des arcs symbolisant les dépendances de données. Une relation directe est maintenue, au cours de la synthèse, entre la représentation algorithmique de ce graphe et sa réalisation structurelle car chacune des opérations possède une correspondance structurelle sous la forme d'un opérateur VLSI particulier puisé dans la base de données : la primitive `max` peut être réalisée par un opérateur de comparaison, par un opérateur spécifique ou encore par un ensemble de portes optimisé manuellement. Les arcs entre les opérations correspondent, quant à eux, à un sous-ensemble des ressources d'interconnexions : registres, multiplexeurs ou bus. Les dépendances de contrôle sont représentées sur cette structure par des nœuds (en pointillé sur la figure 8) qui symbolisent le parallélisme de l'exécution des opérations (au sens de la concomittance d'exécution). Sur cet exemple issu du calcul des gradients, les opérations de soustraction (`sous`) et de valeur absolue (`abs`) sont exécutées en parallèle[4]. Ce nœud de contrôle

4. On peut constater que la valeur absolue est potentiellement exécutable sur le cycle suivant : c'est d'ailleurs une des optimisations possibles durant la synthèse.

réfère d'autre part à la description de la machine à états qui séquencera l'architecture (en indiquant la liste, et les valeurs, des signaux de contrôle à envoyer au chemin de données à cet instant précis).

Les propriétés des graphes de contrôle et de flot de données se résument ainsi :

- il existe une bijection entre un programme fonctionnel et un CDFG (voir les travaux de [37] au sujet de la programmation fonctionnelle) qui accélère d'ailleurs significativement le prétraitement de la spécification,
- au même titre que la représentation fonctionnelle, un CDFG conserve les caractéristiques de lisibilité et de hiérarchie (dues à l'utilisation de macro-fonctions),
- c'est une représentation suffisante des algorithmes et des architectures,
- l'implantation matérielle d'un CDFG est **fonctionnellement équivalente** à la spécification initiale permettant ainsi d'assurer le principe de « validité par construction » de la démarche (cf. 3.2).

C'est donc la **seule** structure de données maintenue durant le processus de synthèse que nous présentons maintenant...

3.6. optimisation globale : le recuit simulé

Les algorithmes d'optimisation sont tous deux basés sur une technique de recherche stochastique : le recuit simulé [43]. Le principe de cet algorithme est le suivant :

1. On dispose d'une solution initiale **viable** (réalisable mais sûrement sous-optimale car arbitraire),
2. On itère un processus qui applique aléatoirement une transformation **locale** et en accepte le résultat avec une probabilité $P(\Delta E)$ (où ΔE est la variation de coût de la solution) jusqu'à atteindre un minimum **global** d'énergie.

Cette probabilité $P(\Delta E)$ s'exprime par une fonction de type Boltzman : $\min(1, \exp -\frac{\Delta E}{\text{Temp}})$. Les paramètres essentiels du recuit simulé sont :

1. L'énergie qui est la fonction de mesure de qualité : ici en termes de surface, de performance, de qualité du contrôle ($\Delta E = \alpha \times \Delta_{\text{surface}} + \beta \times \Delta_{\text{performance}} + \gamma \times \Delta_{\text{controle}}...$)
2. La température qui est le paramètre de contrôle de l'amplitude des transformations. La valeur initiale de ce paramètre est déterminée expérimentalement comme étant celle qui permet d'accepter 50% des premières transformations (empiriquement : $T_{init} = 10 \times \overline{\Delta E}$ avec $\overline{\Delta} = \Delta$ moyen)
3. Les critères d'équilibre local et d'arrêt qui sont définis comme étant le nombre de transformations à appliquer, à température constante, pour atteindre un minimum local d'énergie (pour le critère d'équilibre) et le nombre de transformations infructueuses qui ne génèrent plus de solutions « meilleures »à température faible (pour le critère d'arrêt).

Les différents paramètres de la fonction énergie évoluant dans des proportions généralement différentes, une phase d'exploration aléatoire de l'espace de conception est d'abord nécessaire. Une pondération adéquate de ces termes est effectuée lors de cette étape (typiquement 5000 transformations **toutes acceptées**), conduisant à une expression de la fonction coût de la forme :

$$\Delta E = A\alpha\Delta_{\text{surface}} + B\beta\Delta_{\text{performance}} + C\gamma\Delta_{\text{controle}}$$

où A, B, C sont les critères d'optimisation fixés par le concepteur et α, β, γ les pondérations qui équilibrent l'influence des facteurs telles que :

$$\alpha\overline{\Delta_{\text{surface}}} = \beta\overline{\Delta_{\text{performance}}} = \gamma\overline{\Delta_{\text{controle}}}$$

3.6.1. l'ordonnancement

Cette première étape de synthèse (illustrée figure 9 sur la détection de contours) consiste à affecter à chaque opération d'un CDFG un cycle d'une machine synchrone pour son exécution. C'est également au cours de cet ordonnancement qu'est effectuée la sélection des modules dans la base de données. Les critères de minimisation durant l'ordonnancement (les termes de la fonction d'énergie du recuit simulé) sont la surface du chemin de données (calculée à partir du parallélisme des opérations et de la sélection des modules), le nombre d'états et de cycles nécessaires à l'exécution de l'algorithme complet (ces deux termes peuvent être différents lors de la présence de branchements conditionnels dans l'algorithme) et la complexité du contrôle (anticipée grâce à la mesure originale de régularité des graphes introduite au § 3.7).

Les transformations locales appliquées sur la solution pendant l'ordonnancement sont :

- le déplacement « vers le haut » d'une opération (on avance son instant d'exécution),
- le déplacement « vers le bas »(qui retarde l'exécution),
- le changement de module (on sélectionne un nouveau type de module pour implanter une opération : un soustracteur à la place d'une unité arithmétique et logique par exemple pour implanter la valeur absolue dans le calcul du gradient).

Par la convergence du recuit simulé les opérateurs sélectionnés auront tendance à être en nombre minimal, fonction du compromis surface/contrôle, donc réutilisés le plus possible en moyenne d'où un effet de partage de ressources.

Afin d'optimiser finement le parallélisme des opérations à l'intérieur du graphe, certains séquencements spéciaux sont introduits : le chaînage et le multicyclage d'opérateurs. Le chaînage consiste à effectuer plusieurs opérations en séquence sur le même flot de données à l'intérieur d'un même cycle machine : il apparaît sur le graphe ordonnancé de la figure 9 avec les quatre opérations de la normalisation. Le multicyclage est la transformation inverse qui consiste à mobiliser plusieurs cycles pour l'exécution d'une

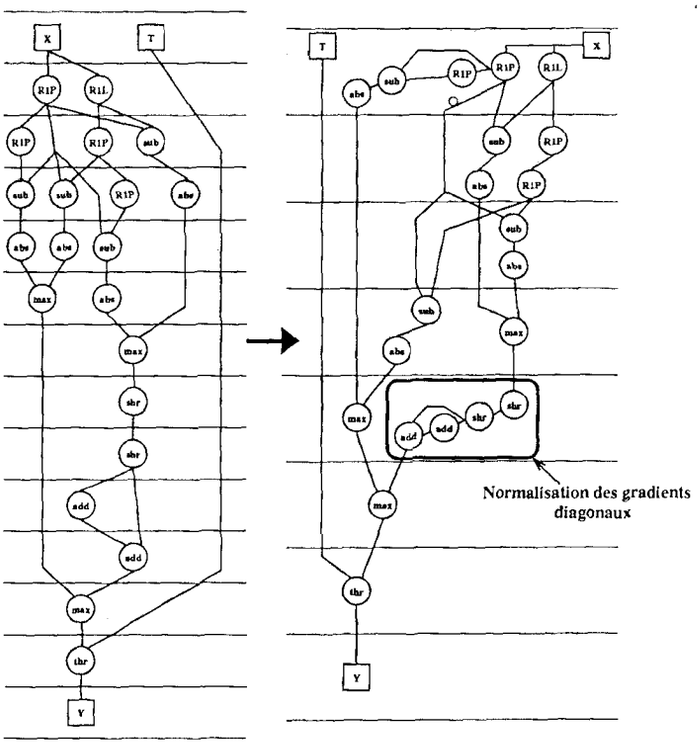


Figure 9. – Effet de l'ordonnancement sur le graphe du détecteur de contours.

opération sur un opérateur« lent » (une multiplication par exemple). Nous verrons au § 3.6.1 que ce séquencement particulier se banalise avec l'usage des macro-fonctions.

La figure 11 indique les évolutions des paramètres pendant l'ordonnancement du détecteur de contours. La variation en exponentielle décroissante par palliers de la température (a) est typique du recuit simulé homogène : la température est décrémentée d'un rapport empirique de 0,95 lorsque l'équilibre thermique local est atteint. On note d'ailleurs à cet égard que la première baisse de température (aux environs de 6.10^4 itérations), pour cause de minimum local atteint, se paie cher en surface (b) pour un gain en performance (d) très momentané. Mais la courbe d'évolution du coût global (la fonction énergie) n'en confirme pas moins la convergence du processus vers un minimum global (e). Les courbes b, c et d illustrent les variations quasi-indépendantes de chacun des paramètres de qualité. Soulignons qu'en l'absence de normalisation préalable, le terme de surface dont l'amplitude moyenne de variation est de l'ordre de plusieurs milliers de μ m^2, serait prépondérant vis-à-vis de la performance et du contrôle.

3.6.2. *l'assignation*

La deuxième optimisation par recuit simulé assigne une instance d'un opérateur particulier (dont le type a été sélectionné au cours de la première étape) à chacune des opérations du graphe. L'assignation est appliquée aussi bien sur les opérations du graphe initial que sur les registres de mémorisation des résultats intermédiaires. On y manipule essentiellement les interconnexions à l'intérieur du chemin de données. La fonction d'énergie est, ici, exprimée en termes de nombre de registres ($\Delta_{\text{registres}}$) et de multiplexeurs ($\Delta_{\text{multiplexeurs}}$) :

$$\Delta E = A\Delta_{\text{multiplexeurs}} + B\Delta_{\text{registres}}$$

Les transformations sont :

- la permutation de deux opérateurs de même type utilisés en parallèle (illustrée figure 10),
- le mouvement de commutativité qui permute les deux entrées d'une opération commutative,
- le mouvement d'assignation qui remplace un opérateur par un deuxième disponible à cet instant.

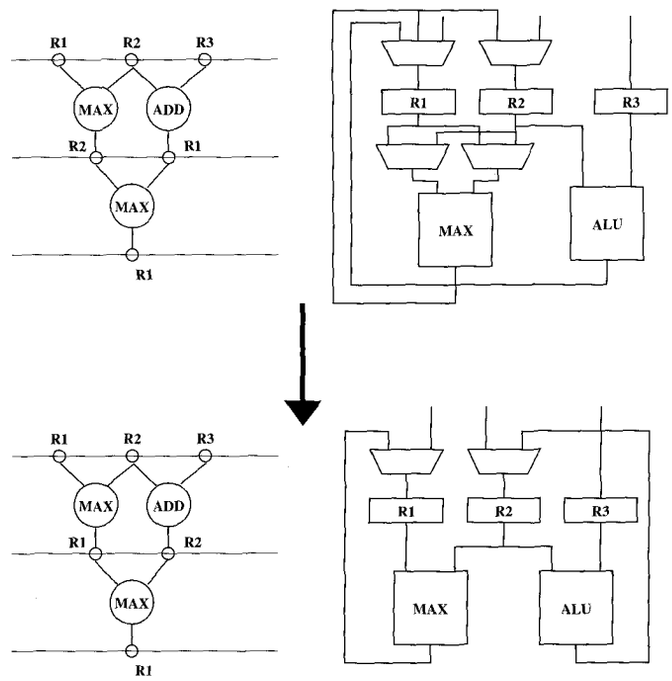


Figure 10. – Effet d'une permutation de deux registres (R1 et R2) sur une partie du chemin de données du détecteur de contours.

Ces transformations ressortent comme étant les plus simples, intuitivement concevables. Toutefois, ce ne sont pas les seules modifications applicables à ce stade de la synthèse. Il est envisageable d'étendre les degrés de liberté de l'optimisation par des transformations plus algorithmiques profitant par exemple de l'associativité de certaines opérations [31] ou encore des transformations architecturales comme la fusion de registres en bancs mémoire ou la réduction de piles de comunications [24]. La figure 10 illustre le gain potentiel d'une transformation élémentaire. Cette permutation des registres R1 et R2 (qui n'affecte pas la sémantique du traitement puisqu'elle ne brise pas de dépendances de données)

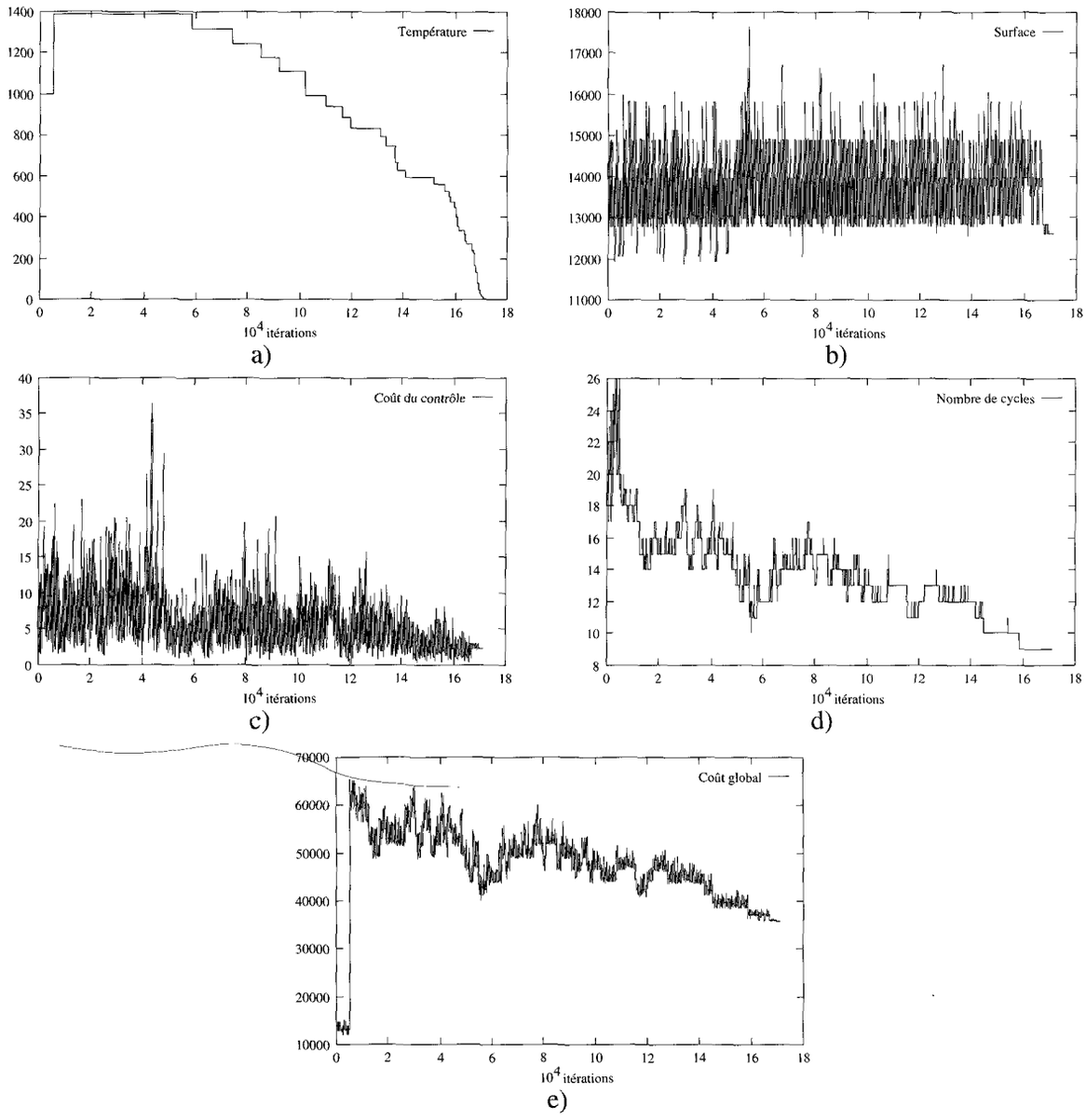


Figure 11. – Évolution des paramètres d'optimisation lors de l'ordonnancement du détecteur de contours.

fait passer le nombre d'entrées de multiplexeurs de 9 à 4 sans modifier le nombre de registres ou d'unités de calcul (respectivement 3 et 2).

La figure 12 retrace les évolutions des paramètres de l'assignation. A proximité d'un optimum, les nombres de registres et de multiplexeurs n'évoluent plus de manière indépendante. L'insertion d'un registre (b) entraîne la plupart du temps la suppression d'un multiplexeur (c) et inversement. Le recuit simulé permet, dans ce cas, de ne pas boucler sur ces configurations considérées équivalentes par rapport aux critères fixés (d).

3.7. optimisation du contrôle : la régularité des graphes

Le développement de cette mesure de régularité répond au problème de l'optimisation globale d'une architecture en termes à la fois de qualité du chemin de données et de simplicité du contrôle [45]. En effet, un contrôle simple est une forte présomption de rapidité accrue par un plus faible nombre de portes à traverser (un chemin critique plus court) pour ne pas évoquer le gain en surface correspondant. Plus précisément, l'objectif est d'optimiser simul-

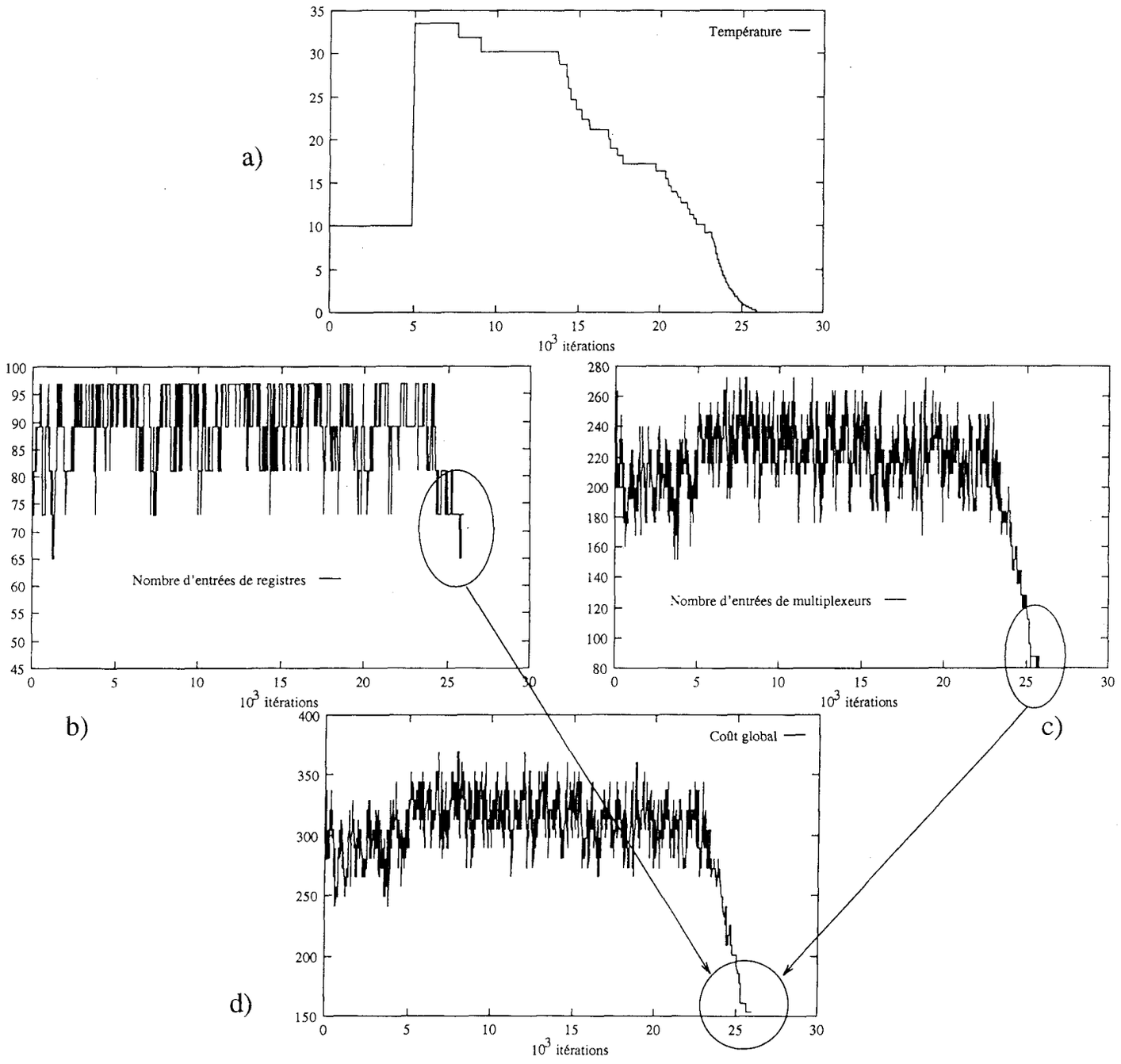


Figure 12. – Évolution des paramètres d'optimisation lors de l'assignation du détecteur de contours.

tanément les parties contrôle et chemin de données d'un circuit. Dans le cas de circuits implantant des traitements d'organisation très régulière (pour lesquels le contrôle est d'ailleurs assez simple : moins de 10% de la surface lui sont consacrés), optimiser la performance de l'unité de séquencement peut conduire à des solutions pour lesquelles le chemin de données devient significativement plus complexe (cf. tableaux 1 et 2). Le compromis entre l'augmentation de surface du chemin de données et le gain en rapidité du contrôleur devient alors difficile à trouver. Mais il n'en va plus de même pour des architectures, notamment de vision, où les procédures impliquent des choix, des séquencements ou des ré-emplois et où la proportion de contrôle est donc importante (dans l'étiqueteur, le contrôle occupe déjà environ 30% du circuit total). C'est pendant la première phase de synthèse (l'ordonnancement) que sont principalement manipulées les informations séquentielles de l'architecture (son contrôle). C'est donc naturellement lors de cette étape que doit se faire l'optimisation de la qualité de sa machine à états.

Or c'est seulement après la deuxième phase de synthèse, celle qui détermine les ressources d'interconnexion, qu'on connaît précisément les séquences de signaux de contrôle et donc qu'on peut mesurer la qualité du contrôleur.

Nous avons alors conjecturé une relation entre une certaine régularité du CDFG et la complexité du contrôle. Cette intuition peut s'appuyer sur l'examen des architectures systoliques à l'intérieur desquelles les transferts de données ainsi que les opérations elles-mêmes sont fortement réguliers et pour lesquelles le contrôle se réduit généralement à une simple distribution de signaux d'horloge. La régularité d'un graphe de contrôle et de flot de données, et par anticipation l'estimation de la complexité de son contrôle, s'exprime alors naturellement comme une **mesure statistique de la diversité des transferts de données**.

Au-delà de cette seule intuition, il y a deux justifications possibles à une telle mesure :

En premier lieu, nous savons que, lors de l'assignation, le nombre de multiplexeurs est minimisé (cf. 3.6.2). Les connexions fixes entre ressources sont ainsi privilégiées. Si d'autre part les transferts de données à l'intérieur d'un graphe sont réguliers (par exemple de durées comparables), dans la description finale de la machine à états qui séquence l'architecture il y aura répétition des mêmes séquences de signaux de contrôle. Or, la compatibilité des signaux de contrôle entre plusieurs transitions d'une machine est une condition importante de sa minimisation en nombre de « termes produits » et donc en surface [3].

Cette justification intuitive *a priori* est complétée, en second lieu, par l'expérience. L'examen de la dynamique d'évolution de la régularité du graphe du détecteur de contours au cours de son ordonnancement, figure 13, montre des variations typiques. On y constate que la régularité varie dans des proportions importantes. Le tableau 1, pour sa part, montre que ces variations sont quelquefois dans les mêmes proportions que celles de la performance ou du nombre d'états, mais que cette mesure n'est corrélée ni avec la surface du chemin de données, ni avec le nombre de cycles de la machine qui implante l'algorithme. Dans ces conditions, la régularité d'un CDFG ne peut pas être dénuée de signification d'une part et, d'autre part, les critères classiques de surface et de performance généralement employés pour optimiser une architecture ne sont pas suffisants pour qualifier complètement cette dernière.

Le calcul de la mesure de régularité est illustré figure 14. Dans un premier temps, les transferts de données du graphe (les motifs) sont identifiés et typés en fonction de leurs opérations source et destination (i, j). Chaque motif possède une caractéristique de longueur (le nombre de cycles « traversés » par le motif). La régularité du graphe est ensuite calculée sur une matrice d'interconnexions dont les éléments $Mat_{i,j}$ sont des couples $(\sum_{i,j}, \sigma_{i,j})$ où $\sum_{i,j}$ est la moyenne et $\sigma_{i,j}$ l'écart-type des longueurs des motifs de type (i, j). La diversité totale du graphe est une somme pondérée de tous les termes $\sigma_{i,j}$, la régularité étant l'inverse de cette diversité.

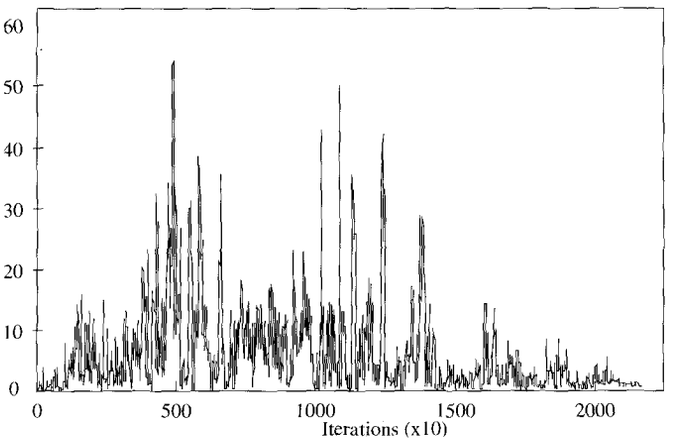


Figure 13. – Évolution classique de la régularité d'un graphe en cours d'optimisation.

Le tableau 1 donne les résultats expérimentaux de l'impact de la minimisation de la diversité d'un graphe de flot de données (diversité initiale σ_i, diversité finale σ_f) sur la densité des interconnexions (nombre de registres et de multiplexeurs Γ_T), sur la surface (S_C) et le délai (D_C) du contrôleur[5], et sur la surface du chemin de données (S_D) après compilation du circuit, ainsi que les gains en surface de contrôle (Δ_{S_C}) et surface de chemin de données (Δ_{S_D}) pour quatre exemples d'algorithmes synthétisés. `conv33` est une simple convolution 3×3, `ell5` est le filtre elliptique du cinquième ordre, classique étalon dans ce domaine, `cont` est le détecteur de contours, et `ecc8bits` est la partie flot de données de l'algorithme d'étiquetage en composantes connexes.

Tableau 1. – Impact de la minimisation de la régularité sur les surfaces et délais des contrôleurs et chemin de données de quelques circuits synthétisés. Les surfaces sont exprimées en nombre de portes, les délais en nanosecondes

graphe	σ_i	σ_f	Γ_T	S_C	D_C	Δ_{S_C}	S_D	Δ_{S_D}
conv33	0,88	5,44	232	551	14,9		5415	
	0,88	0,50	240	532	14,4	-3,5%	5684	+4,9%
cont	1.50	6,19	145	251	11,8		3775	
	1.50	2,42	121	225	12,5	-10%	3801	+0,7%
ell5	29,71	107,84	672	2044	18,4		7422	
	29,71	6,32	656	1789	16,7	-12%	7419	-0,04%
ecc8bits	3,68	6,54	57	693	14,8		1551	
	3,68	1,23	61	555	11,9	-20%	1722	+11%

Dans les quatre cas, la surface du contrôleur S_C est significativement réduite (de 3,5% à 20%). En revanche, cette réduction en surface du contrôleur se traduit dans deux cas (`conv33`et `ecc8bits`) par une augmentation du nombre de registres ou de multiplexeurs Γ_T d'où une surface de chemin de données S_D qui augmente (de 4,9% et 11% respectivement).

5. Le délai du circuit dans notre modèle architectural est voisin de celui du contrôle effectué en parallèle, la densité des interconnexions introduisant des délais comparables.

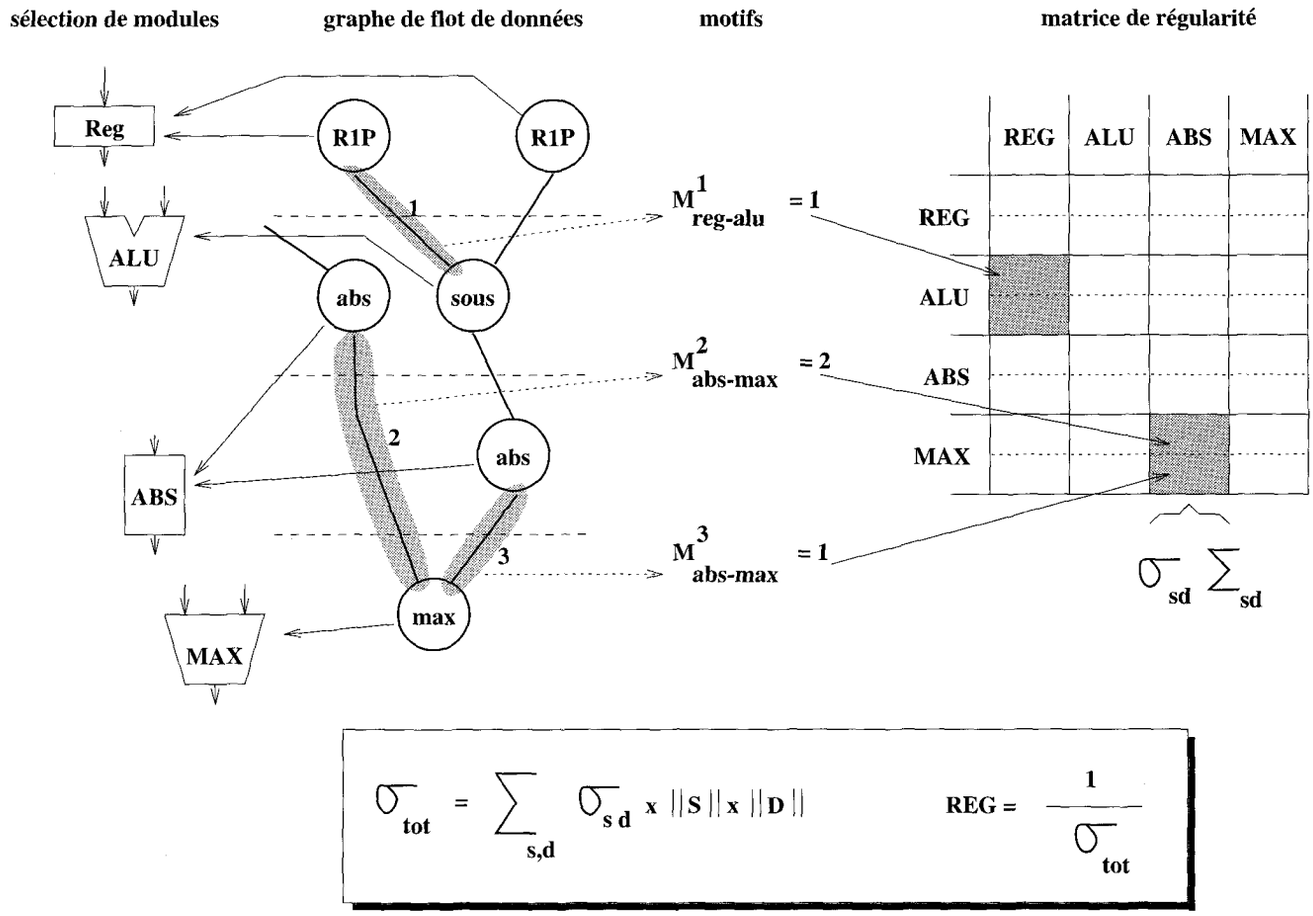


Figure 14. – Calcul de la régularité du CDFG du détecteur de contours.

Tableau 2. – Comparaison des surfaces des circuits réguliers et non-réguliers.

graphe	version	Surface totale	Δ_S
conv33	irrégulière	5966	
	régulière	6216	+4%
cont	irrégulière	4026	
	régulière	4026	0%
ell5	irrégulière	9466	
	régulière	9208	-2,8%
ecc8bits	irrégulière	2244	
	régulière	2277	+1,4%

Dans tous les cas, on constate que l'usage et l'optimisation de la régularité des CDFG anticipe, comme prévu, la complexité du contrôle en termes à la fois de surface (nombre de portes donc coût du silicium) mais également de rapidité. Les surfaces totales des quatre circuits synthétisés sont données sur le tableau 2. Dans les cas où le contrôleur occupe une surface importante de silicium (`ell5` et `ecc8bits` avec des rapports contrôle/chemin de données de 25% et 40% respectivement), la surface totale du circuit est réduite de 2,8% ou légèrement augmentée (à cause de l'augmentation en surface de chemin de données). A l'inverse, lorsque la surface du contrôleur est de proportion réduite (`cont` et `conv33` avec des rapports de 6% et 10% environ) l'impact de la régularité n'est pas significatif sur la surface totale du circuit. Une expertise de plus haut niveau (éventuellement humaine de la part du concepteur) apparaît donc nécessaire pour qualifier les gains respectifs en surface et performance des circuits et opter, le cas échéant, pour une solution irrégulière mais moins coûteuse en surface.

3.8. résultats

Le tableau 3 donne les caractéristiques du circuit de détection de contours réalisé avec l'outil de CAO COMPASS dont le layout est donné figure 15.

L'étape de synthèse en elle-même (de la spécification FP à la représentation RTL en VHDL) a nécessité environ 50 secondes CPU sur une station de travail de type SUN SPARC-II alors que

la phase de compilation du *silicium* dans l'environnement COMPASS, en tenant compte du placement et du routage des blocs RTL, de la vérification des horloges et des plots d'entrée/sortie peut prendre plus d'une journée : la plupart de ces étapes ne sont en effet pas automatiques et requièrent une intervention humaine significative.

Etant donné l'état de développement expérimental actuel du système ALPHA, la synthèse est également une opération qui reste dans une grande partie interactive : choix des pondérations de surface, performance et régularité, paramétrage des processus de recuit simulé et vérification de la cohérence du code VHDL. Le développement du logiciel ALPHA a été fait en langage C et rassemble plus de 20000 lignes de code qu'il est toujours difficile de tester entièrement.

Tableau 3. – Caractéristiques du circuit d'extraction de contours après placement et routage

Surface totale	14,6 mm^2
Nombre de transistors	15626
Surface du contrôleur	0,22 mm^2
Surface du chemin de données	6 mm^2
Cellule RAM précompilée	1,98 mm^2
Nombre de cycles	8
Diversité du graphe	2,4212
Nombre d'entrées de multiplexeurs	56
Nombre d'entrées de registres	73
Fréquence du circuit	22 Mhz
Fréquence des pixels	2,78 Mhz

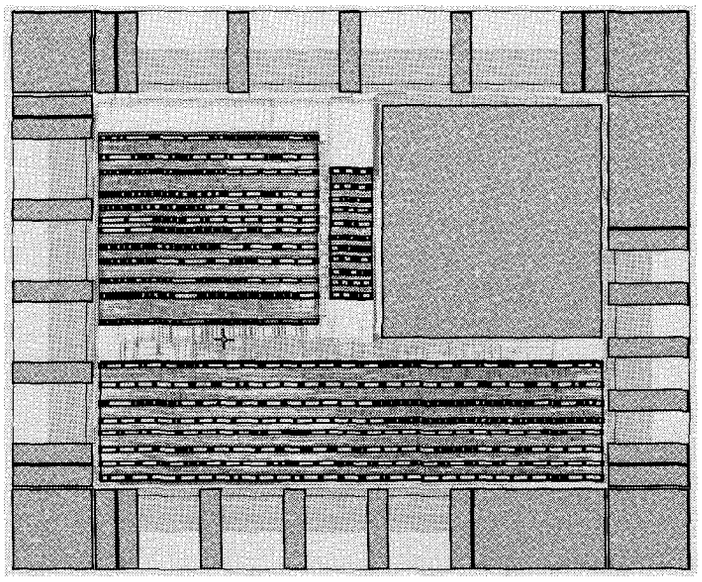


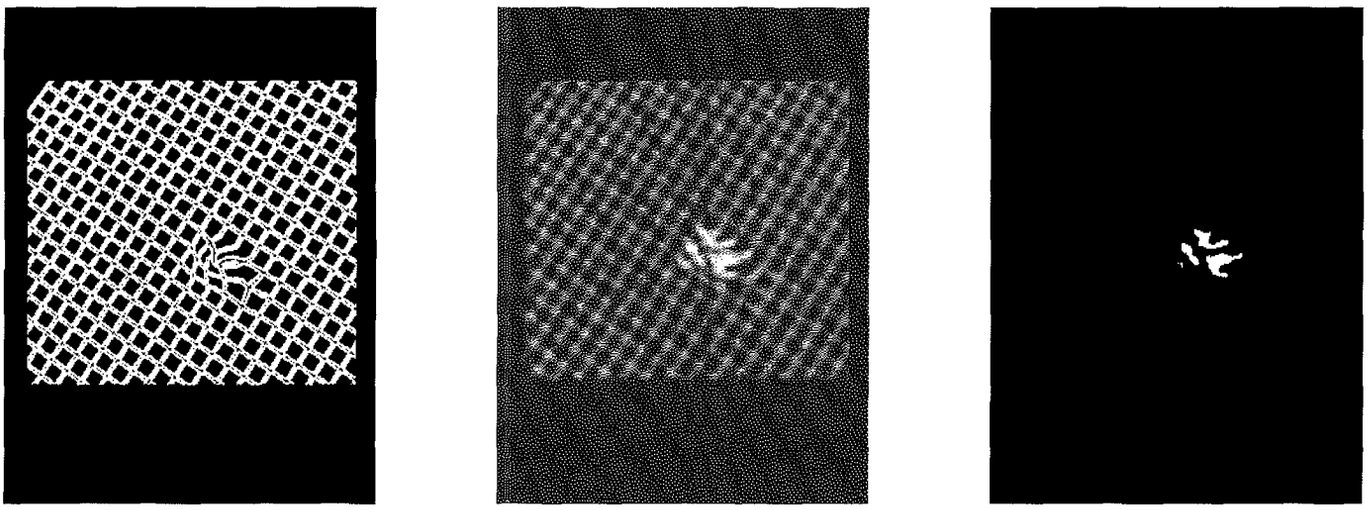
Figure 15. – Layout du circuit d'extraction de contours synthétisé avec ALPHA puis compilé avec COMPASS.

4. vers les systèmes visuels intégrés : exemple d'un détecteur de défauts

On est désormais en mesure de réaliser automatiquement un circuit de traitement d'images comme un filtre ou le détecteur de contours précédent, tout en garantissant qu'il fonctionnera correctement au sein du système pour lequel il a été conçu. Mais deux remarques s'imposent :

- un automate de vision, aussi simple soit-il, demande l'organisation de plusieurs opérateurs de traitement d'images et au moins une décision, quelque dix fois plus de transistors qu'une simple détection locale (cf. chapitre 1). Sachant que la puissance d'émulation est bornée, de même que celle de l'ordinateur qui supporte optimisation et conception (l'outil COMPASS nécessite au moins 256 Mo de mémoire pour fonctionner), et que la demande ne peut que croître, est-il raisonnable de prétendre confier à ALPHA, en un seul bloc, la conception du système perceptif d'une automobile par exemple où se côtoieraient détection de lignes, poursuite couleur et analyse de région ([15, 16], [42])?
- l'équilibrage des différentes variables de coût associé au modèle d'automate choisi fait que sans autres précautions la performance du système conçu peut chuter, car en proportions on perd plus en surface qu'on ne gagne en rapidité (cf. Amdahl [2] par exemple). Partant d'un détecteur de contours opérationnel à la volée sur le Calculateur Fonctionnel, on aboutit à un circuit qui ne fonctionnerait plus que six fois moins vite (voir la performance tableau 3). Or la réunion de plusieurs fonctions ne peut que ralentir ([2] à nouveau et [21]) le système complet par rapport à la puissance totale impliquée, d'où d'ailleurs l'importance croissante du contrôle avec le caractère « système ». Il est donc critique de montrer, en une sorte de relaxation sur les contraintes, que l'on peut rediminuer *a posteriori* le temps de réponse, et mesurer l'impact de cette nouvelle option sur la surface.

C'est pourquoi cette dernière partie traite de deux techniques indépendantes dans l'état actuel, qui complètent le logiciel pour une synthèse de systèmes plus complexes : la synthèse hiérarchique et la synthèse d'architectures pipelines. La première a pour objectif d'améliorer le niveau sémantique des algorithmes par extension de la bibliothèque des opérateurs (primitives) disponibles. La seconde retravaille la performance des architectures construites. Un point commun entre ces méthodes et avec ce qui précède est, là encore, l'exploitation d'une certaine forme de régularité : la synthèse hiérarchique (présentée au § 4.2) s'intéresse à la régularité à la fois sémantique d'un algorithme (par l'identification de



(a) Image originale (b) Image des distances (c) Image résultat

Figure 16. – Détection d'un défaut sur une image fortement texturée.

macro-fonctions) et structurelle de l'architecture (par l'implantation répétée de ces macro-fonctions comme macro-cellules) tandis que la méthode de synthèse pipeline (§ 4.3) s'intéresse à et exploite la régularité temporelle d'une architecture exécutant un algorithme (par découpage du CDFG correspondant).

L'algorithme qui sert de support ici, ressortit à la vision préattentive. C'est un type d'automate de vision parmi les plus simples que l'on puisse concevoir, pour des fonctions d'alerte notamment en environnement particulier (restreint aux textures à dominante périodique). Il n'en utilise pas moins quatre opérateurs de contours globalement comparables[6] au précédent et a conduit à un circuit dont la complexité est déjà de l'ordre de 10^5 transistors pour, rappelons-le, une fonctionnalité vision encore limitée.

4.1. présentation de l'algorithme

Cet algorithme est basé sur un modèle de vision humaine inspiré des travaux de [8]. Il peut se décomposer en trois parties :

- Calcul des points de contour, avec direction. Ce calcul est fait par passage dans une table de transcodage (LUT) des contributions dx et dy de chaque pixel en valeurs d'arcs (la LUT contient $arctg(dx, dy)$). L'image des contours résulte de la détection des points dont les dérivées spatiales sont significatives (seuillage du maximum de dx et dy).
- pour un certain nombre d'orientations, k, (nous avons volontairement restreint ce nombre à 4) régulièrement réparties sur le cercle, on mesure dans tous les voisinages 3x3 les directions des contours, puis un écart moyen des distances de ces directions par rapport à chaque valeur k ainsi que l'écart moyen global sur l'image,
- la détection d'un défaut correspond à un maximum local significatif de toutes les contributions.

Un résultat de cet algorithme sur image texturée et déformée est représenté figure 17.

4.2. spécification et synthèse hiérarchique

L'identification d'une certaine forme de régularité sémantique dans l'algorithme est naturelle sur cet exemple : répétition de la partie de l'algorithme qui calcule les contributions respectives des quatre directions traitées. Le code FP résultant de ce découpage est listé figure 17.

Le processus de synthèse se déroule de la façon suivante :

- optimisation et synthèse de la macro-fonction de traitement d'une direction `dirk`,
- compilation sous COMPASS de la macro-cellule correspondante, extraction de ses caractéristiques technologiques (surface précise, délai),
- ajout de la cellule dans la bibliothèque VHDL des modules VLSI de ALPHA avec les informations extraites du layout (nombre de portes, nombre de cycles d'horloge, fréquence maximale de fonctionnement,...),
- Optimisation et synthèse du circuit complet en utilisant la macro-cellule comme un module VLSI classique. Lors de cette étape, certaines contraintes sont ajoutées comme un temps de cycle minimum (celui correspondant au temps de cycle de la macro-cellule) et la longueur du nœud multicyclé.

6. Amplitude et direction du vecteur gradient ne sont pas exploitées dans les mêmes conditions, aux mêmes stades des algorithmes.

```
// source FP du détecteur de défauts

MAIN [
INPUT Image:PIXEL;
INPUT Thresh:PIXEL;
OUTPUT Res:PIXEL;
]
def dx = sous . [Image, Pixel_delay . Image];
def dy = sous . [Image, Trame_delay . Image];

def angle = xor . [sgn . dx, sgn . dy];
def adx = abs . dx;
def ady = abs . dy;
def m = max . [adx, ady];

def tg1 = atg_lut . [adx, ady];
def tg = norm . [tg1, angle];

def edge = thr . [Thresh, m];
def direction = select . [tg, edge];

def d0 = dir_macro(dir = 0) [direction, edge];
def d1 = dir_macro(dir = 1) . [direction, edge];
def d2 = dir_macro(dir = 2) . [direction, edge];
def d3 = dir_macro(dir = 3) . [direction, edge];

def Res = max ! [d0, d1, d2, d3];
END
```

```
// source FP de la macro de direction

MAIN [
INPUT dir:PIXEL;
INPUT edge:PIXEL;
OUTPUT out:PIXEL; ]

def ex=gauss(center = param).dir;

def eT = Trame_delay . ck;
def eTT = Trame_delay . eT;
def sT  = add \ [ex, eT, eTT];
def eP = Pixel_delay . sT;
def ePP = Pixel_delay . eP;
def ed = add \ [sT, eP, ePP];

def a1 = select .  [ed, edge];
def c1 = count16(outwidth = 8) . edge;
def c2 = accu(outwidth = 16) . a1;
def a2 = div(outwidth = 8) . [c2, c1];
def out = div . [a1, a2];
END
```

Figure 17. – Spécification hiérarchique du détecteur de défauts.

Comme prévu au § 3.6.1, la macro-fonction ne s'exécute en effet pas en un seul cycle.

Le tableau 4 résume les caractéristiques des deux sessions de synthèse (macro-fonction `dirk` et circuit complet). Dans les deux cas, comme supposé en conclusion du chapitre 3, la complexité des algorithmes ne permet pas une exécution en temps réel sur des séquences d'images de taille raisonnable (fréquence pixel maximale de 1,3 Mhz pour le circuit complet), sauf à retravailler les performances des circuits.

Le dessin du circuit synthétisé est donné figure 18. Il est facile d'identifier les quatre instances de la macro-cellule aux quatre coins.

Tableau 4. – Caractéristiques de la macro-cellule et du circuit de détection de *défauts complet*

	macro de direction	détecteur de défauts
Émulé sur	59 processeurs	294 processeurs
Technologie	1 μm	1 μm
Surface	12 mm^2	82 mm^2 (circuit)
Nombre de transistors	18116	83685
Cycles-machine	10	23
Fréquence pixel	3 Mhz	1.3 Mhz
Fréquence d'horloge	30 Mhz	30 Mhz

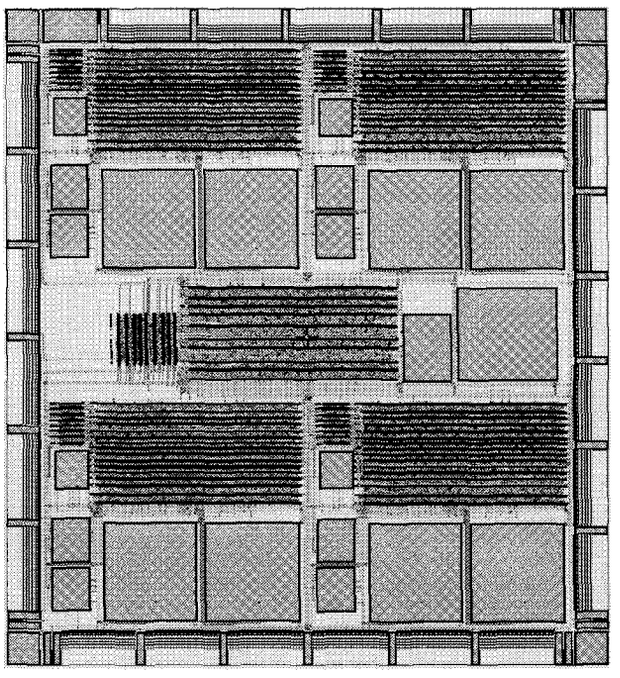


Figure 18. – Layout du circuit d'extraction de défauts.

4.3. synthèse d'architectures pipelines

Lors de la synthèse d'une architecture classique (non-pipeline), l'hypothèse de fonctionnement est que le délai qui sépare deux données distinctes à traiter est supérieur au temps de calcul du circuit. En traitement d'image, le débit moyen des données à traiter ainsi que la complexité des algorithmes de traitement conduisent généralement à des solutions qui ne sont pas réalisables en version non-pipeline. Dans ce cas, les données arrivent dans le circuit sous forme de « vagues » séparées par des intervalles de quelques cycles seulement.

La figure 19 illustre le détail de fonctionnement d'un algorithme exécuté en mode pipeline. Sur cette figure, les points représentent les données qui « transitent » dans l'algorithme. Le Délai d'Introduction des Données (DID) correspond à deux cycles de la machine et la durée du traitement complet est de 6 cycles. Il y a donc en permanence neuf données traitées simultanément dans l'architecture à des degrés divers de la procédure (sur des cycles algorithmiques différents). Pour se représenter les différentes opérations du même algorithme exécutées simultanément sur des données distinctes, on décide de fusionner sur un même pseudo-cycle toutes les opérations traitant des données aux mêmes instants.

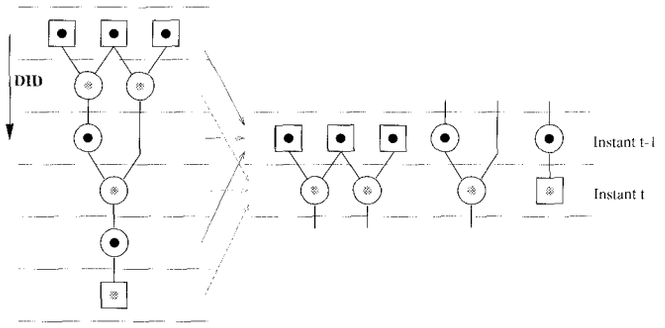
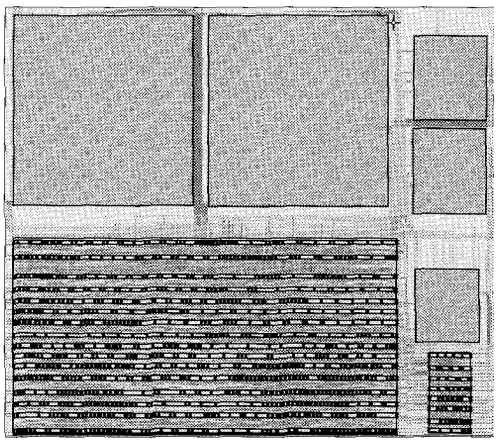


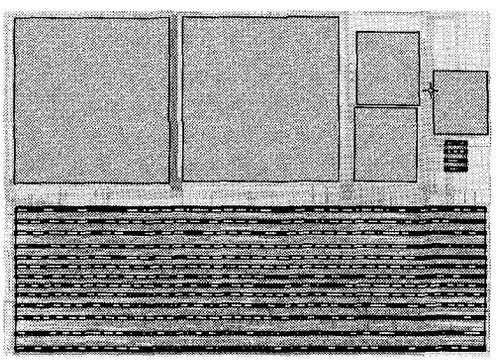
Figure 19. – Séquencement pipeline d'un algorithme de traitement : lorsque trois flots successifs de données sont traités par l'algorithme (à deux cycles d'intervalle), les opérations traitant des données simultanément sont vues en parallèle. Sans transformation aucune du graphe initial, on peut se représenter la version pipeline de l'architecture en liant les cycles mis en parallèle. Cette façon de « voir » le graphe permet de synthétiser indifféremment des architectures pipeline ou non.

La différence entre la synthèse d'une architecture classique et d'une architecture pipeline consiste justement en cette façon de se représenter le graphe de flot de données. Dans le cas pipeline, un graphe dit *composite* est créé par la mise en parallèle des cycles-machine distants de la valeur du DID. Ce graphe composite n'existe que par cette nouvelle manière de « regarder » le CDFG initial. Aucune structure nouvelle n'est construite et toutes les transformations et mesures valables dans le cas non pipeline sont conservées ici. Toutefois, la sémantique temporelle des transferts de données entre les opérations est changée : un transfert à travers plusieurs niveaux d'un pipeline « transporte » plusieurs données consécutives. La régularité du CDFG à laquelle nous avons fait correspondre la complexité du contrôle perd une partie de sa signification. Le contrôleur d'une architecture pipeline possède d'ailleurs beaucoup moins d'états mais plus de signaux de contrôle (par la mise en parallèle de plusieurs modules) et sa complexité est plus fortement dépendante du nombre d'étages de pipeline que de la régularité du CDFG elle-même.

La figure 20 donne les dessins des deux versions (classique et pipeline) de la même macro-fonction `dirk`. Leurs caractéristiques sont résumées dans le tableau 5. La différence en surface des contrôleurs s'explique par le fait que le contrôleur de la version pipeline, bien qu'il possède un nombre de signaux plus élevé (à cause du plus grand parallélisme des opérateurs), ne possède que deux états! En revanche, le chemin de données possède un plus grand nombre de modules dans la version pipeline. La différence n'est, cependant, pas aussi significative qu'on pourrait le supposer à cause d'un certain nombre de modules qui ne peuvent pas être partagés, version pipeline ou non (une FIFO de retard ligne et quelques mémoires LUT). Les caractéristiques de performance de la version pipeline (15,4 Mhz) autorisent cette fois-ci une exécution en temps réel de la fonction sur des séquence d'images de taille 780×780.



(a) Version non pipeline



(b) Version pipeline

Figure 20. – Les versions pipeline et non pipeline de la macro-fonction de traitement d'une direction (les échelles relatives de représentation ont été conservées).

Tableau 5. – Caractéristiques des versions pipeline et non pipeline de la même fonction. On ne tient pas compte dans ce tableau des surfaces des opérateurs FIFO et LUT/ROM, cellules précaractérisées et présentes dans les deux circuits (d'où la différence entre 12 mm^2 du tableau précédent et 4,56 mm^2 (4,36 + 0,2) ici).

	version non pipeline	version pipeline
Surface du chemin de données	4,36 mm^2	5,26 mm^2
Surface du contrôleur	0,2 mm^2	0,06 mm^2
Nombre de cycles ou latence	10	2
Fréquence d'horloge du circuit	30,7 Mhz	30,7 Mhz
Fréquence des données	3,08 Mhz	15,4 Mhz

5. conclusion

Nous avons présenté dans cet article les résultats de nos travaux sur la synthèse automatique des architectures de vision dérivées d'une émulation sur une machine dédiée aux applications en temps réel. Nous avons montré en particulier l'intérêt et la faisabilité du concept d'émulation en temps réel des algortithmes de traitement d'images dans un but non seulement de prototypage rapide des applications (développement et mise au point des algorithmes) mais aussi d'expérimentation de solutions intégrées. Dans ce cadre, le logiciel de synthèse ALPHA est capable, dès aujourd'hui, de concevoir des architectures VLSI embarquables et ce, avec un degré d'automatisation relativement avancé. Même si l'intervention d'un architecte de circuit est, et sera longtemps encore, indispensable pendant la conception, nous montrons avec cet environnement, malgré son état d'avancement expérimental, qu'une grande partie de la conception de haut niveau est à la portée d'un système automatique.

Nous expliquons comment une certaine forme de régularité des applications peut être avantageusement estimée puis exploitée lors de la construction automatique d'une architecture dédiée. Optimiser la régularité d'un graphe CDFG permet de construire des architectures dont le contrôle, et donc le séquencement et la performance, sont meilleurs. L'estimation de la régularité que nous proposons autorise cette optimisation du contrôle au stade précoce de la synthèse où elle est la plus efficace alors que les informations pertinentes ne sont pas *a priori* disponibles. D'autre part, les régularités sémantique, structurelle et temporelle des applications ont été largement utilisées pour concevoir des architectures très probablement proches de l'optimalité : c'est-à-dire qui respectent les contraintes de performance qu'implique le temps réel tout en étant minimales au sens de leur surface et globalement « simples », donc rapides, à concevoir. Plus généralement, nous avons adopté une démarche expérimentale pour identifier et comprendre les problèmes de la synthèse : c'est, avant tout, cela que nous voulons exposer ici. Car une fois l'ensemble du processus bien maîtrisé, la synthèse automatique de circuits peut s'avérer plus rationnelle pour l'étude des systèmes autonomes que la classique programmation d'essai sur ordinateur, éventuellement suivie d'une transplantation laborieuse. Nous en apportons des éléments de preuve sous la forme d'un système qui, illustré ici au travers de trois exemples significatifs par leur progression en complexité, préfigure les outils de conception automatique de haut niveau déjà indispensables et bientôt opérationnels.

BIBLIOGRAPHIE

[1] E. Allart and B. Zavidovique. Functional image processing through implementation of regular data flow graphs. In *Proceedings of 21st Annual Asilomar Conference on signals, systems and computers. Pacific Grove, CA USA*, November 1987, pp. 705–708.

[2] G. M. Amdahl. Validity of the simple processor approach to acheive large scale computing capabilities. In *Proceedings AFIPS, vol 30*, 1967.

[3] Pranav Ashar, Srinivas Devadas, and A. Richard Newton. *Sequential Logic Synthesis*. Kluwer Academic Publishers, Norwell, Massachusetts, USA, 1992.

[4] P.M. Athanas and A.L. Abbott. Real-Time Image Processing on a Custom Computing Platform. *Computer*, 28(2) : Fév. 1995, pp. 16–24.

[5] J. Backus . Can programming be liberated from the Von-Neumann style? A functional style and its algebra of programs. *Communications of the ACM*, 1978, 21.

[6] M. Barbacci. Instruction set processor specifications (ISPS) : The notation and its applications. In *IEEE Trans. on Computers*, volume C-30, January 1981, pp. 24–40.

[7] Patricia Bournai, Bruno Chéron, Thierry Gautier, Bernard Houssais, and Paul Le Guernic. SIGNAL Manual. Technical Report 745, IRISA/INRIA Rennes, July 1993.

[8] Virginia Brecher. New techniques for patterned wafer inspection based on a model of human preattentive vision. *SPIE Journal on Applications of Artificial Intelligence : Machine Vision and Robotics*, 1708 : 1992, pp. 452–459.

[9] R. Camposano. Structural synthesis in the Yorktown Silicon Compiler. In *Proceedings of VLSI'87, Vancouver*, August 1987.

[10] Raul Camposano and Wayne Wolf, editors. *High-Level VLSI Synthesis*. Kluwer Academic Publishers, 1991.

[11] J. Canny. A computational approach to edge detection. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 8(6) : 1986, pp. 679–698.

[12] S. Dacic, Th. Bommart, and B. Zavidovique. A friendly interface for complex machine programming. In *Proceedings of COMPINT'87, Montreal, CA*, November 1987.

[13] Serge Dacic. *Conception et exploitation intuitive de systèmes informatiques complexes*. Thèse de doctorat, Université de Paris XI Centre d'Orsay, 1990.

[14] R. Deriche. Fast algorithms for low level vision. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 12(1) : 1990, pp. 78–86.

[15] E. D. Dickmanns. *Active Vision*, chapter Expectation-based dynamic scene understanding, pages 303–335. MIT Press, 1992.

[16] E. D. Dickmanns. Machine perception exploiting high–level spatio–temporal models. In *Machine perception (La perception de l'environnement par senseurs automatiques)*. AGARD Lecture Series 185 (Advisory Group for Aerospace Research and Development). OTAN, Neuilly-Sur-Seine, FRANCE, 1992.

[17] Richard O. Duda and Peter E. Hart. *Pattern Classification and Scene Analysis*. Wiley-Interscience, New-York, 1973.

[18] E.F. Girczyc and J.P. Knight. An ADA to standard cell hardware compiler based on graph grammars and scheduling. In *Proceedings. of ICCD*, 1984, pp. 726–731.

[19] G. Goossens, D. Lanneer, J. Vanhoof, J. Rabaey, J. Van Meerbergen, and H. De Man. Optimisation-based synthesis of multiprocessor chips for digital signal processing, with CATHEDRAL-II. In *Proceedings of the International Workshop on Logic and Architecture Synthesis for Silicon Compilers, Grenoble, FRANCE*, 1988.

[20] Pravil Gupta, Chih-Tung Chen, J. C. DeSouza-Batista, and Alice C. Parker. Experience with image compression chip design using Unified System Construction tools. In *Proceedings of the 31st ACM/IEEE Design Automation Conference, CD-ROM Publication, San Diego, CA, USA*, 1994.

[21] J. L. Gustafson. Re-evaluating Amdahl's law. *Communications of the ACM*, 31(5) : 1988, pp. 532–533.

[22] Dave Johannsen. Bristle Blocks : a silicon compiler. In *Proceedings of the 16th ACM/IEEE Design Automation Conference*, 1979.

[23] David W. Knapp and Alice C. Parker. A unified representation for design information. In *Proceedings of the 7th International Symposium on Computer Hardware Description Languages and their Applications*, August 1985, pp. 337–353.

[24] I.C. Kraljić, G.M. Quénot, and B. Zavidovique. From Real-Time Emulation to ASIC Integration for Image Processing Applications. In *Proc. of the 8th Annual IEEE International ASIC Conference*, pages 31–4, Sept. 1995. Austin, TX, USA.

[25] Jean Mermet, editor. *VHDL for Simulation, Synthesis and Formal Proofs of Hardware*. Kluwer Academic Publishers, 1992.

[26] J. W. Modestino and R. W. Fries. Edge detection in noisy images using recursive digital filtering. *Computer Graphics and Image Processing*, 6 : 1977, pp. 409–433.

[27] John K. Ousterhout, Gordon T. Hamachi, Robert N. Mayo, Walter S. Scott, and George S. Taylor. Magic : a VLSI layout system. In *Proceedings of the 21st ACM/IEEE Design Automation Conference, Albuquerque, New Mexico, USA*, 1984, pp. 152–159.

[28] L. Palmier, C. Legrand, F. Devos, and B. Zavidovique. A functional operation architecture for image processing . In *International Symposium on Circuits and Systems*, June 1985. Kyoto, Japan.

[29] Luc Palmier. *Conception fonctionnelle de circuits intégrés de traitement d'images*. Thèse de troisième cycle, Université de Paris XI Centre d'Orsay, 1985.

[30] Pierre G. Paulin. Algorithms for high-level synthesis with area and interconnect constraints. In *Proceedings of the EURO ASIC'89, Grenoble, France*, 1989, pp. 144–158.

[31] Miodrag Potkonjak and Jan Rabaey. Optimizing resource utilization using transformations. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, CAD-13 : 1994, pp. 277–292.

[32] Stephane Praud. *Implantation d'un algorithme d'étiquetage en composantes connexes sur le Calculateur Fonctionnel*. Rapport de DEA, Université de Paris Sud Centre d'Orsay, 1993.

Manuscrit reçu le 20 avril 1995

[33] J. Rasure. KHOROS : Visual language and software development for image processing. *International Journal on Imaging Systems and Technologies*, 1990, pp. 183–199.

[34] A. Safir and B. Zavidovique. Towards a global solution to high-level synthesis problems. In *Proceedings of European Design Automation Conference. Glasgow, Scotland, U.K.*, March 1990, pp. 283–288.

[35] A. Safir and B. Zavidovique. A solution to the high level synthesis problems. *The International Journal of Computer Aided VLSI Design*, 3 : January 1991, pp. 43–69.

[36] G. Saucier and J. Trilhe, editors. *IFIP Transactions on Synthesis for Control Dominated Circuits*, volume A-22. North-Holland, 1992.

[37] J. Sérot, G.M. Quénot, and B. Zavidovique. Functional programming on a data-flow architecture : applications in real-time image processing. In *International Journal of Machine Vision and Applications*, volume 7, 1993, pp. 44–56.

[38] J. Sérot, G.M. Quénot, and B. Zavidovique. Calculateur Fonctionnel : une architecture flot de données pour le traitement d'images en temps réel. *Revue Scientifique et Technique de la Défense*, 1994, pp. 217–229.

[39] Jocelyn Sérot. *Mise en œuvre d'un formalisme fonctionnel pour la programmation d'une architecture flot de données dédiée au traitement d'images temps réel*. Thèse de doctorat, Université de Paris Sud Centre d'Orsay, 1993.

[40] Jay R. Southard. MacPitts : An approach to silicon compilation. *COMPUTER*, pages 74–82, December 1983.

[41] D.E. Thomas, E.D. Lagnese, R.A. Walker, J.A. Nestor, J.V. Rajan, and R.L. Blackburn. *Algorithmic and Register-Transfer Level Synthesis : The System Architect's Workbench*. Kluwer Academic Publishers, Norwell, Massachusetts, USA, 1990.

[42] M. A. Turk, D. G. Morgenthaler, K. D. Gremban, and M. Marra. Vits – a vision system for autonomous land vehicle navigation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 10(3) : 1988, pp. 342–361.

[43] P.J.M. van Laarhoven and E.H.L. Arts. *Simulated Annealing : Theory and Applications*. D.Reidel Publishing Company. Eindhoven, 1988.

[44] F. Verdier and B. Zavidovique. A complete environment for global architecture synthesis. In *Proceedings of Computer Architectures for Machine Perception Workshop, New Orleans, USA*, December 1993, pp. 77–81.

[45] François Verdier. *Conception de logiciels d'optimisation sous contraintes d'architectures VLSI pour le traitement d'images : le problème du contrôle*. Thèse de doctorat, Université de Paris XI Centre d'Orsay, 1995.

[46] Naeem Zafar. Using emulation to cut ASIC and system verification time. *ASIC Design*, April 1994.

[47] B. Zavidovique, C. Fortunel, Quénot G., A. Safir, J. Sérot, and F. Verdier. Automatic synthesis of vision automata. In Magdi A. Bayoumi, editor, *VLSI Design Methodologies for Digital Signal Processing Architectures*, pages 261–318. Kluwer Academic Publisher, 1994.

[48] B. Zavidovique, V. Serfaty, and C. Fortunel. Mechanism to capture and communicate image-processing expertise. *IEEE Software*, November 1991, pp. 37–50.

LES AUTEURS

François VERDIER

est docteur de l'université de Paris-XI en Électronique depuis Janvier 1995. Il a effectué ses recherches au sein du laboratoire Système de Perception de l'Établissement Technique Central de l'Armement à Arcueil sur le thème de la synthèse automatique d'architectures de vision embarquées. Il enseigne aujourd'hui à l'Université de Cergy (Val d'Oise). Ses domaines d'intérêt sont principalement axés sur les descriptions comportementales des algorithmes et des architectures, la synthèse automatique, la vérification formelle et les techniques d'optimisation des architectures VLSI. Il est membre IEEE depuis 1995.

Bertrand ZAVIDOVIQUE

est professeur a l'Université Paris XI (Institut d'Electronique Fondamentale) . Conseiller de l'ETCA/DRET pour les problèmes Temps réel et Robotique, il y a fondé le laboratoire « Systeme de Perception ». Ses recherches intéressent la conception de Systèmes de Perception, l'impact de leur organisation interne sur leur efficacité externe, leur implantation effective. Elles sont concretisées par 5 brevets (rétines programmables, calculateur fonctionnel, dispositif robotisé multicapteurs) et plus de deux cents publications scientifiques en Traitement d'Image, Perception des robots, Architectures spécialisées, Implantation VLSI, Contrôle symbolique et Apprentissage, par la construction effective de plusieurs machines informatiques dont certaines industrialisées, par l'écriture d'un environnement logiciel de programmation en T.I. distribué sous licence et par la méthode de synthèse automatique d'architectures integrées illustrée dans cet article.