Amélioration de la fiabilité des calculateurs parallèles SIMD par test et tolérance aux fautes structurelle

F. Clermidy

INSTITUT NATIONAL POLYTECHNIQUE DE GRENOBLE

N° attribué par la bibliothèque

THESE

Pour obtenir le grade de

DOCTEUR DE L'INPG

Spécialité : microélectronique

Préparé au laboratoire DTA/DEIN/SLA du CEA Saclay

Dans le cadre de l'école doctorale microélectronique

Présentée et soutenue publiquement

Par

Fabien Clermidy

Le 8 décembre 1999

Titre :

amélioration de la fiabilité des calculateurs parallèles SIMD par test et tolérance aux fautes structurelle

Directeur de thèse :

M. Mihalis Nicolaïdis, directeur de recherche à l'INPG/TIMA

JURY

M. Daniel Litaize, Professeur à l'IRIT (Université Paul Sabatier) , Président
M. Christian Landrault, Directeur de recherche au LIRMM , Rapporteur
M. Stanislaw Piestrack, Professeur des universités de Pologne , Rapporteur
M. Eric Martin, Professeur du LESTER (université de Bretagne sud)
M. Mihalis Nicolaïdis, Directeur de recherche au TIMA , Directeur de recherche
M. Thierry Collette, Docteur-ingénieur au CEA , Responsable de thèse CEA

Avant-propos

Les travaux présentés dans ce mémoire ont été réalisés au sein du Service Logiciel et Architecture (SLA) du Département d'Electronique et d'Instrumentation Nucléaire (DEIN) du Commissariat à l'Energie Atomique (CEA) de Saclay. Je tiens ici à remercier les responsables de m'avoir accueilli et donné les moyens pour accomplir mon travail de recherche : Monsieur Jean-Baptiste Thomas, chef du DEIN, et son successeur Monsieur François Pupat, ainsi que Monsieur Didier Juvin, chef du SLA.

Je remercie Monsieur Litaize Daniel, professeur à l'université Paul Sabatier, de m'avoir fait l'honneur de présider mon jury de thèse.

Messieurs Christian Landrault, directeur de recherche au LIRMM, et Stanislaw Piestrack, professeur des universités de Pologne, sont les rapporteurs de cette thèse. Je tiens à les remercier pour l'intérêt qu'ils ont porté à mes travaux.

Je tiens également à remercier Messieurs Eric Martin, professeur à l'université de Bretagne sud, et Serge Cruzel, ingénieur en sûreté de fonctionnement pour leur participation à mon jury de thèse.

Monsieur Michel Nicolaïdis, directeur de recherche à l'INPG, a été mon directeur de thèse pendant ces 3 années. Je le remercie pour ses nombreuses remarques concernant mon travail. Sa connaissance parfaite du domaine du test m'a été d'un précieux concours.

Enfin, Monsieur Thierry Collette a été, pendant ces 3 années, celui qui a suivi au jour le jour l'avancée de mes travaux. Au delà de ses compétences techniques concernant les architectures, il a su m'écouter tout au long de mon travail, et a toujours été présent dans les moments difficiles. Je tiens ici à saluer ses qualités humaines, et la confiance sans faille qu'il m'a toujours accordée.

Je me tourne maintenant vers ceux qui ont dû me « supporter » pendant ces années. Plus particulièrement, je tiens à citer :

Jean-François Larue, responsable du groupe architectures parallèles, pour son accueil et le soutien qu'il a apporté à mes travaux ; les gens du « hard », Alexa Depeyrot dont l'aide fut précieuse pour la réalisation de l'ASIC de SYNTOL et que je remercie également pour les divers services qu'elle m'a gentiment rendu, Renaud Schmit, toujours à l'aise avec les cartes électroniques et qui a supervisé la réalisation de la carte prototype ; Philippe Fauvel alias FIFO, le roi des travaux en tout genre ; Christian Gamrat, avec qui la conversation est toujours passionnante, que ce soit sur les architectures reconfigurables, les réseaux de neurones ou l'Afrique ; et enfin Francis Dupin, mon voisin de bureau avec qui j'ai partagé quelques soucis communs ; Côté « soft », la sympathique équipe d'imagerie dans toutes les

dimensions (parfois un peu bruyante...), Marc Viala et son franc-parler, Sylvie Naudet, qui n'a pas son pareil pour s'énerver devant un PC, Patrick Sayd, compagnon de foots endiablés ; l'équipe des crayons et pastilles, Laurent Letellier qui sait réchauffer l'atmosphère, Patrick hède, que je tiens à remercier ici pour son intérêt vif pour mon travail et pour sa relecture de ce mémoire avec un œil extérieur, Delphine Popot dont l'humeur joviale n'est jamais démentie ; et enfin l'équipe de Stretch, Hassane, Michel et Marc dont les vastes connaissances m'ont appris beaucoup sur les calculateurs parallèles et leurs langages.

Un merci tout particulier à Evelyne Parfenoff, Mireille Plonquet et Mireille Agus qui ont su m'aider à surmonter dans la bonne humeur les tâches administratives.

Trois stagiaires ont participé activement à la réalisation du prototype. Sans eux, celui-ci n'aurait jamais pu voir le jour. Je tiens donc à remercier Daniel Payrard pour sa contribution au placement et routage de l'ASIC SYNTOL, Emilie Raffard pour ces développements en VHDL de programmes fonctionnels pour l'ASIC, Frédéric Blanc pour la conception de la carte prototype fournissant l'environnement de test nécessaire à l'ASIC et pour sa contribution au simulateur permettant les calculs de fiabilité. Ces travaux me furent particulièrement précieux.

Comme ce sont les thésards qui ont toujours su le mieux comprendre les autres thésards, je voudrais citer ici ceux qui m'ont accompagné au moins pendant un bout de route : Marc et Anne, Jean-Michel, Patrick, mon voisin de bureau, ami dont les conseils avisés m'ont beaucoup servi notamment dans les moments difficiles, Cédric qui m'a fait découvrir les théâtres parisiens et qui est devenu le roi de la course à pied, Seb le triathlète passionné, Corinne et ses bases de données colorées, et Fred le petit dernier.

Les amis servent également à assurer l'équilibre d'un thésard. Je voudrais donc ici les remercier : William, Yves et Ludo, les lyonnais, Gérard, Valérie, Dav', Valérie, Tosh, P'tit Bras, Laure, le petit Pierre, Hans et Bab, les valenciennois.

Une pensée toute particulière va à Sophie, mon soutien quotidien, lectrice acharnée de ce mémoire, impitoyable face aux mauvaises tournures de phrase et aux explications vaseuses.

Enfin, il y a ceux qui me soutiennent depuis des années, à commencer par mes parents dont je tiens à rendre hommage ici pour leur confiance, Sophie et Pascal, Muriel et Lionel, et ceux qui font ma joie, Emma ma douce filleule, Célia et Loan.

Je dédie ce manuscrit à la mémoire de Jean-Luc.

Avertissement

Les domaines de l'électronique et de la micro-électronique fourmillent d'expressions et de mots anglais. Certains de ces mots sont passés dans le langage courant de l'homme de métier. Ils seront utilisés tels quels dans ce mémoire. D'autres mots possèdent une traduction qui n'est pas usitée dans la profession. Dans ce cas, le mot anglais sera également conservé.

Par ailleurs, le mot reconfiguration n'est pas correct en français. Ce néologisme sera néanmoins utilisé tout au long de ce mémoire, car il permet d'alléger le texte sans aucune perte en compréhension.

Sommaire

Introduction

Le domaine de la micro-électronique est en progrès constant et nul ne doute que la célèbre loi de Moore continuera à être suivie pendant ces dix prochaines années. On voit ainsi se profiler des circuits intégrés au milliard de transistors. Concevoir un tel circuit entraîne des questions sur l'utilisation qui peut être faite de la puissance disponible. Une simple extension des architectures de circuits actuels ne peut pas répondre à ces questions, car le temps de développement nécessaire serait énorme. Il devient donc crucial d'imaginer de nouvelles architectures ayant une ou plusieurs des caractéristiques suivantes :

- fort pourcentage de ré-utilisation de circuits antérieurement écrits ou réalisés par une source extérieure ;
- duplication de certains blocs ;
- plus d'indépendance entre les blocs du circuit. Cela est dû au temps de propagation d'un bout à l'autre d'un ASIC qui devient prépondérant sur le temps de commutation des transistors ;
- organisation stricte de la couche physique (layout), afin de minimiser l'importance des longueurs d'interconnexions ;
- hiérarchisation du circuit pour faciliter le contrôle.

Les calculateurs parallèles, constitués de processeurs identiques, possèdent naturellement un grand nombre des qualités d'une architecture future. Parmi ceux-ci, les calculateurs SIMD (Single Instruction Stream, Multiple Data Stream) permettent de traiter un grand nombre de données dans un même cycle. Ils allient la simplicité à l'efficacité et peuvent être avantageusement alliés à des processeurs séquentiels (calculateurs multi-SIMD) afin d'améliorer leur souplesse d'utilisation. Leurs applications vont du traitement d'image au calcul intensif en passant par les calculs de corrélation (sur une base de donnée par exemple). Leur fréquence de fonctionnement dépend alors de leur intégration, c'est à dire de la proximité physique des processeurs. On voit alors facilement l'avantage qu'on pourrait tirer en les intégrant sur un seul circuit intégré de type ASIC (Application Specific Integrated Circuit).

Dans ce mémoire, nous traitons des calculateurs parallèles SIMD et multi-SIMD, dont la caractéristique principale est une forte intégration. Un tel niveau d'intégration n'est pas sans soulever des problèmes, parmi lesquels la gestion de la température : cette dernière, si elle est mal maîtrisée, peut mettre à mal la fiabilité de la structure. Par ailleurs, les problèmes de fiabilité dus à l'insertion rapide de nouvelles technologies (non parfaitement stabilisées) sont un des challenges que doit relever l'industrie des semi-conducteurs. Les calculateurs parallèles fortement intégrés sont alors confrontés à un double risque de perte de fiabilité : un premier au niveau du circuit intégré, et le second au niveau de la structure même qui inclut un grand nombre de ces circuits intégrés. Il était donc tout à fait logique de se pencher sur ce problème qui peut, à terme, empêcher la réalisation physique d'une telle structure.

Les buts de fiabilité que nous nous fixons sont de deux ordres :

- assurer avec une forte probabilité que le calculateur parallèle pourra fonctionner pendant plusieurs années sans nécessiter d'arrêt de maintenance ;
- assurer, avec une bonne probabilité, que le résultat donné par le calculateur est correct.

Un moyen pour augmenter la fiabilité d'un système est la tolérance aux fautes. Celle-ci consiste à empêcher qu'un défaut de la structure ne "dégénère" en panne. Les approches de tolérance aux fautes sont nombreuses mais suivent tous la même idée principale : l'utilisation de redondance logicielle ou matérielle, en temps ou en information, doit permettre de découvrir une manifestation du défaut.

Dans ce mémoire, nous présentons une solution de tolérance aux fautes adaptée aux calculateurs parallèles fortement intégrés. Celle-ci est appelée tolérance aux fautes structurelles et est basée sur la notion de reconfiguration. Cette dernière consiste à conserver la topologie du réseau d'interconnexion du calculateur, même en présence de fautes. A la fin de ce mémoire, nous donnons des éléments de réponse aux deux questions "quelle tolérance aux fautes pour les calculateurs parallèles fortement intégrés ?" et "quel niveau de fiabilité peut-on atteindre ?".

Dans notre approche, l'amélioration de la fiabilité est étroitement liée avec le test du calculateur. Ce dernier n'est actuellement pas toujours bien intégré lors de l'étape de conception de la structure. Par ailleurs, il est soumis à de profondes modifications dues aux nouvelles technologies. Il nous a donc semblé important de traiter le problème du test des calculateurs SIMD ou multi-SIMD comme une part entière de ce mémoire.

Ainsi, nous proposons une chaîne complète menant à une amélioration de la fiabilité de calculateurs SIMD et multi-SIMD. Deux éléments de cette chaîne sont étudiés : il s'agit du test de fabrication et lors du fonctionnement, qui est présenté dans les chapitres 2 et 3, et de la reconfiguration des calculateurs, qui est présentée dans les chapitres 4, 5 et 6. Le chapitre qui suit cette introduction sert, quant à lui, à définir précisément le contexte dans lequel nous nous sommes placé.

Chapitre 1 : présentation générale

1 Introduction

Avec l'avancée des évolutions technologiques, il apparaît de plus en plus nécessaire de ne plus séparer la théorie de l'architecture des circuits et ses moyens de réalisation pratique. Une composante importante liée à la réalisation d'un circuit intégré est le test. Les ingénieurs de conception s'accordent en général pour dire que celui-ci peut représenter jusqu'à 60 % du temps total consacré à la réalisation d'un circuit intégré ou d'un système électronique. En effet, les techniques de test sont nécessaires afin de détecter les erreurs de conception (cahier des charges mal défini, transcription imprécise...) et de valider le circuit en fin de fabrication et pendant son fonctionnement.

La fiabilité des circuits et des systèmes est également liée à toute réalisation physique. Elle est liée à une probabilité de bon fonctionnement, c'est à dire de respect des spécifications d'origine pendant un temps de fonctionnement important. Le problème de la fiabilité fait partie d'une classe plus importante : celle de la sûreté de fonctionnement (figure 1). Les fondements de cette dernière ont été donnés par [Laprie1985], et les définitions de ses principales caractéristiques sont rappelées en Annexe 1.

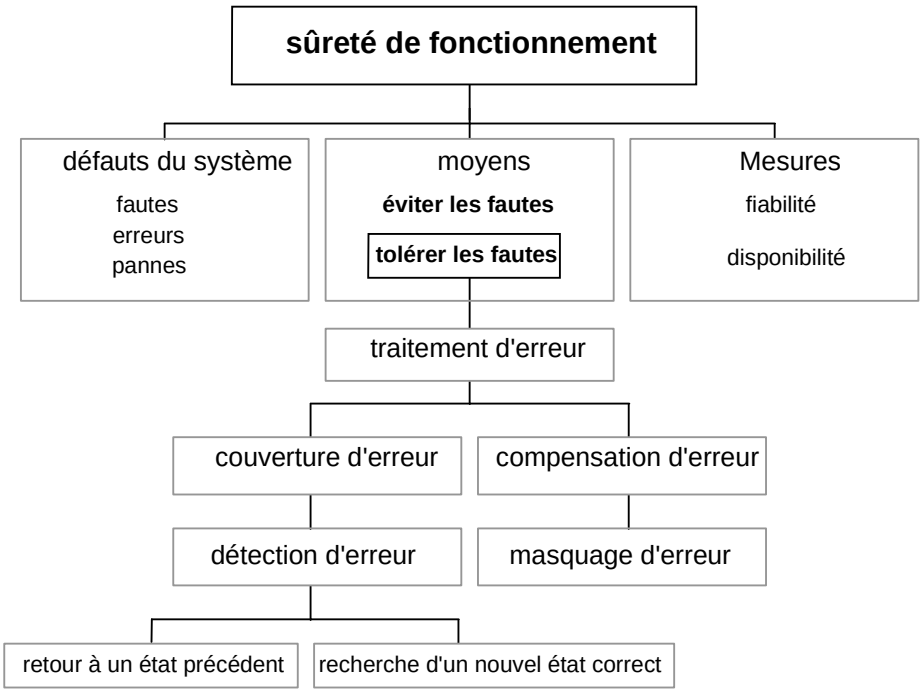


Figure 1 : les termes de la sûreté de fonctionnement en 3 classes (les défauts du système, les moyens d'améliorer la fiabilité et les mesures de la fiabilité)

Les calculateurs parallèles, formés de centaines de processeurs identiques, « noyés » dans une structure globale, posent des problèmes spécifiques aux niveaux du test et de la fiabilité. D'une part, les moyens de test sont souvent délicats à mettre en œuvre à cause du caractère « enfoui » des processeurs, donc difficiles à contrôler. D'autre part, les calculateurs parallèles, considérés comme des accélérateurs puissants de calculs, se doivent de fonctionner 24 heures sur 24. Une intervention de maintenance, lorsqu'elle est possible, ne doit pas impliquer l'arrêt de ces structures pendant de longues heures. Cela signifie que le calculateur doit posséder ses propres capacités de test (on parle alors d'auto-test) et de réparation (auto-repair) quand une erreur du système est détectée. La correction de ces erreurs doit en outre être invisible pour l'utilisateur.

Les calculateurs parallèles possèdent toutefois des avantages importants en termes de test et de fiabilité. Ainsi, leur structure hiérarchique peut être utilisée de façon à faciliter les procédures de test. De même, la redondance intrinsèque de leurs processeurs facilite l'intégration de méthodes permettant d'améliorer la fiabilité telle que la tolérance aux fautes (cf. Annexe 1).

Dans ce mémoire, nous nous basons sur une approche pragmatique de la fiabilité. Les calculateurs parallèles ne sont pas considérés comme étant critiques, comme peuvent l'être ceux utilisés dans l'avionique ou pour les contrôles de trafic. De ce fait, l'amélioration de la fiabilité ne doit pas se faire aux dépens d'une diminution sensible des performances. Cependant, la méthode mise en œuvre doit permettre d'atteindre un niveau de fiabilité élevé quant à la continuité du service que délivre le calculateur (disponibilité) et la justesse des résultats fournis.

Dans notre approche, les moyens utilisés pour le test doivent également aider à l'augmentation de la fiabilité de la structure. Par ailleurs, le test doit être parfaitement adapté aux calculateurs parallèles : les blocs ajoutés pour le mettre en œuvre ne doivent pas détériorer les performances. Malgré ces contraintes, le test doit permettre d'obtenir une bonne couverture des défauts de la structure, ce qui reste son principal objectif.

Avant d'appréhender l'approche globale mise en oeuvre dans ce mémoire (paragraphe 4), nous présentons les éléments nécessaires à la compréhension du problème. Nous commençons par décrire les architectures parallèles. Nous essayons ensuite de donner quelques indications sur la fiabilité attendue d'un processeur et d'un calculateur parallèle n'ayant pas de moyens pour augmenter leur fiabilité.

2 Les calculateurs parallèles

2.1 Introduction

Dans les ordinateurs à architecture traditionnelle dite de Von Neumann, toutes les opérations sont effectuées de manière séquentielle. Les instructions sont d'abord lues, puis décodées. Lorsque cela est nécessaire, les opérandes sont également recherchés. L'opération est alors exécutée puis les résultats mémorisés. Aucune de ces opérations ne peut être lancée avant que l'opération précédente ne soit finie. Cette architecture a servi à la conception des premiers processeurs.

Les concepteurs se sont alors efforcés, tout au long des années, d'augmenter la puissance de calcul des structures. Bien entendu, l'intégration de plus en plus de transistors sur une même surface (voire dans un volume) permet des améliorations importantes, mais souvent jugées insuffisantes. De nouveaux concepts ou des algorithmes plus puissants peuvent alors rendre les calculs plus rapides (par exemple, l'algorithme de Booth pour la multiplication [Hayes1988]). Lorsque tout cela est insuffisant, l'augmentation de la puissance de calcul ne peut être obtenue que par une modification profonde de l'architecture. Celle-ci est guidée par un seul but : celui de paralléliser les actions. Les méthodes pour y parvenir sont par contre complètement disparates, et peuvent aller d'une parallélisation au niveau d'un programme jusqu'à celle des opérations les plus élémentaires. Ainsi, toute architecture de calcul moderne intègre à un niveau ou un autre une forme de parallélisme.

Dans ce contexte, le terme de calculateur parallèle est difficile à définir. Il est donc intéressant de classer les calculateurs selon leur forme et leur niveau de parallélisme. Les taxonomies les plus couramment utilisées sont celles de Flynn [Flynn1972] et de Ducan [Ducan1990], présentées maintenant.

2.2 Taxonomie de Flynn

La classification de Flynn [Flynn1972] repose sur la connaissance des flots d'instruction (I) et des flots de données (D) qui composent le calculateur. Elle distingue donc 4 types de structures, SISD, MISD, SIMD et MIMD :

- les architectures dites à flots d'instructions et de données uniques (SISD : Single Instruction stream, Single Data stream) ;

Ce sont en général les processeurs séquentiels d'architecture Von Neumann classique. Aucune architecture de processeur moderne ne répond à cette stricte définition. On peut alors choisir d'inclure dans cette classe les architectures ayant un degré de parallélisme « faible », de bas niveau ou très localisé.

C'est le cas, par exemple, de la multiplication des unités de calcul ou de l'emboîtement des traitements sur les instructions et les données (pipeline). Les processeurs du commerce, de jeu d'instructions x86 comme le Pentium, ou les processeurs RISC (Reduced Instruction Set Computer) comme l'Alpha21164, le powerPC, le MIPS ou le SPARC rentrent alors dans cette catégorie.

Par ailleurs, certains processeurs utilisent des « vecteurs » de données sur lesquels se font les opérations (processeurs vectoriels). Avec les évolutions du jeu d'instructions et de l'architecture de ces processeurs vers un parallélisme de plus haut niveau, la frontière des architectures dites SISD est de plus en plus ténue. Cela montre les limites de la taxonomie de Flynn.

- les architectures à flot d'instructions unique et à flots de données multiples (SIMD : Single Instruction stream, Multiple Data stream) ;

Dans cette classe d'architectures, plusieurs processeurs exécutent simultanément la même instruction, issue d'une seule Unité de Contrôle (UC), sur des flots de données différents (figure 2a).

Chaque processeur possède alors une mémoire locale qui lui permet de stocker les résultats intermédiaires.

Les machines CM-2 [CM1991] et MPP [Batcher1980] sont deux des représentants les plus connus de cette classe. Le LETI-DEIN a développé 2 calculateurs appartenant à cette catégorie : Sympati II [Juvin1988] en collaboration avec l'IRIT et Symphonie [Collette1998] développé en collaboration avec la SAT. Les calculateurs SIMD ont été plus ou moins abandonnés au début des années 90 en raison de leur manque de souplesse, sauf pour certaines applications comme le traitement d'images. Néanmoins, le fait d'avoir des flots séparés pour traiter les données reste une solution efficace pour le calcul rapide sur un grand nombre de données. Ainsi, des extensions dites SIMD sont ajoutées à la plupart des nouveaux processeurs. Par conséquent, l'association d'architectures SIMD à un ensemble de processeurs SISD (processeurs séquentiels, DSP,...) semble être une solution d'avenir [Peythieux1998]. De telles architectures sont appelées multi-SIMD (figure 2b).

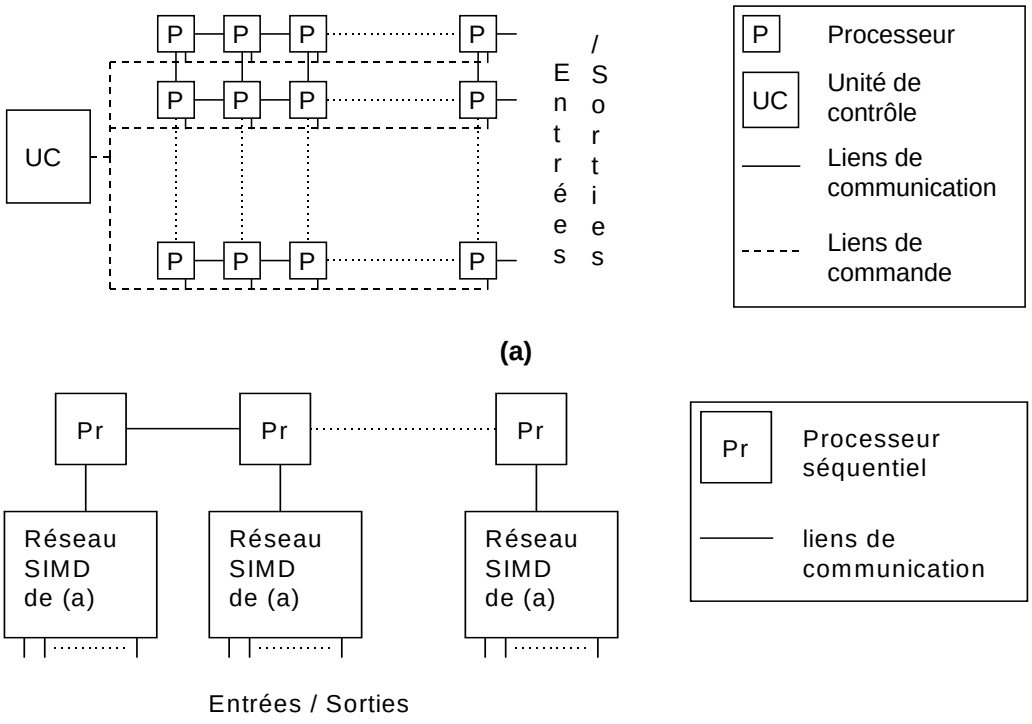


Figure 2 : (a) architecture SIMD, (b) exemple d'architecture multi-SIMD

- les architectures à flots d'instructions multiples et à flot de donnée unique (MISD : Multiple Instruction stream, Single Data stream) ;

Pour ces structures, une séquence de données est envoyée à un premier processeur qui exécute une instruction avant de l'envoyer vers le processeur suivant. C'est donc un système d'emboîtement des traitements (pipeline) de processeurs. Le calculateur systolique est un exemple d'architecture MISD qui permet de résoudre les problèmes de baisse de performance dus aux limitations des

entrées/sorties de toute structure. Ces calculateurs sont toutefois difficiles à mettre en œuvre et manquent de souplesse. Par conséquent, il en existe assez peu, et ils sont souvent dédiés à des applications particulières.

- les architectures à flots d'instructions et de données multiples (MIMD : Multiple Instruction stream, Multiple Data stream) ;

Dans ces structures, les processeurs effectuent simultanément des instructions différentes sur des données différentes. Ils sont également souvent appelés multiprocesseurs (par abus de langage). Cette forme de parallélisme est celle qui a été le plus largement étudiée et implémentée. On peut citer, à titre d'exemple, le premier Cray-1 et les tous derniers calculateurs massivement parallèles à usage général (comme les ASCII red et blue) en passant par les ordinateurs CrayT3D et T3E de SGI, la CM5 [CM1992] de TMC et le Ncube [Palmer1988]. Tous les supercalculateurs actuels utilisent un modèle MIMD plus ou moins respectueux de la définition générale (notamment au niveau de la programmation de ces systèmes).

Les calculateurs MIMD sont souvent classés en deux catégories : les multiprocesseurs à couplage fort (ou à mémoire partagée), où tous les processeurs de tailles comparables se partagent l'accès à une mémoire commune (définition de Enslow), et les multiprocesseurs à couplages faibles (ou à mémoire distribuée) où les mémoires sont locales [Enslow1977]. Enfin, ces multiprocesseurs fonctionnent en parallèle à différents niveaux hiérarchiques de programmation. Ceux-ci peuvent aller du programme (le plus haut niveau) jusqu'à l'instruction qui est l'unité élémentaire d'exécution, en passant par la tâche, et la procédure [Tabak1991].

2.3 Taxonomie de Ducan

La taxonomie de Flynn ne permet pas de classer certaines architectures de façon tout à fait cohérente et elle est souvent dépassée car les distinctions entre calculateurs deviennent de plus en plus subtiles. Nous avons vu que les processeurs du commerce intègrent de plus en plus de fonctionnalités dites SIMD, comme l'extension MMX du Pentium [Peleg and Weiser1996]. Pour palier cette difficulté, Ducan propose une autre taxonomie basée sur le caractère synchrone ou non du calculateur [Ducan1990] : un calculateur synchrone ne possède qu'une horloge globale et un contrôleur de programme. On distingue alors trois classes de calculateurs :

- les calculateurs synchrones. On retrouve dans cette classe les processeurs vectoriels et pipelines, ainsi que les calculateurs SIMD et systoliques ;
- les calculateurs asynchrones. Ce sont les structures MIMD à couplage fort ou faible ;
- les calculateurs mixtes. Ils nous permettent d'introduire les calculateurs Multi-SIMD, comme l'ILLIAC IV de l'université de l'Illinois. Dans cette classe, il faut également ajouter les calculateurs reconfigurables dont la structure peut à loisir se comporter comme une structure SIMD, MIMD ou Multi-SIMD (par exemple, PASM de l'université de Purdue). Nous pouvons encore citer les architectures dataflow, contrôlées par les données elles-mêmes, et les architectures « wavefront array », mélanges d'architectures systoliques et dataflow.

2.4 Conclusion

Les deux taxonomies de Flynn et Ducan permettent de faire un rapide tour d'horizon des calculateurs parallèles. La prépondérance des calculateurs MIMD pour les super-calculateurs d'utilisation générale ne doit pas pour autant faire oublier les autres formes de parallélisme. En effet, ces structures sont toujours énergivores et très coûteuses, avec un encombrement important. Lorsque les puissances de calcul doivent être importantes et/ou que ces calculateurs doivent être embarqués, il est nécessaire de revenir à des calculateurs plus simples, comme les tableaux SIMD. Par exemple, le calculateur Symphonie [Collette1998], développé par le CEA-LETI, fait partie du système de veille infrarouge du Rafale.

Nos travaux se basent sur des architectures de type SIMD ou Multi-SIMD qui permettent d'atteindre des puissances de calcul élevées pour un faible encombrement. Celles-ci ont fait l'objet de travaux antérieurs au sein du laboratoire [Collette1993, Peythieux1998].

Les évolutions technologiques influent sur celles des architectures. Il paraît donc nécessaire d'en prendre compte afin de comprendre l'approche suivie. Le paragraphe suivant y est consacré.

3 Evolutions technologiques et conséquences

3.1 Introduction

Le secteur de l'électronique est en constant progrès et la course en direction des nanotechnologies se poursuit à bon train. Dans son « road map » de 1994 et sa révision de 1998, l'association de l'industrie des semi-conducteurs [SIA1994] prévoit l'évolution technologique jusqu'en 2010. Selon ces prédictions, en cette année 2010, l'industrie sera en mesure de fournir des processeurs dont les caractéristiques principales seraient :

- plus de 800 millions de transistors ;
- quelques milliers d'entrées/sorties;
- des bus de 1000 bits ;
- et une fréquence d'horloge de 2 GHz.

La consommation d'un tel circuit serait alors de 180 W crête, ce qui laisse présager quelques problèmes de refroidissement.

Ce qui suit présente tout d'abord les implications de ces évolutions technologiques en termes de fiabilité. Puis, dans une seconde partie, nous évoquons quelques problèmes liés aux nécessaires évolutions architecturales.

3.2 Evolutions technologiques et fiabilité

Jusqu'à présent, les modèles utilisés pour les circuits électroniques sont des lois purement déterministes. Mais, on sait parfaitement depuis le début du siècle, avec l'avènement de la physique atomique, que plus on se rapproche de la taille de l'atome, plus les lois qui servent à modéliser les

phénomènes sont d'ordre probabiliste. Même si la taille de l'atome (le nanomètre) n'est pas encore atteinte, les « portes logiques » utilisées dans les toutes dernières technologies ne comportent déjà plus qu'un nombre relativement restreint d'atomes. Les chercheurs commencent donc à se pencher sur les comportements atomiques des futures technologies. A l'extrême, nous pouvons dire que le processeur du futur comportera des portes logiques qui délivreront un signal S avec une probabilité P(S). A ce stade, nous parlerons de nanotechnologies et nous devrons revoir complètement notre mode de pensée pour la conception des circuits intégrés ainsi que celui de l'électronique dans son ensemble.

Même si nous n'en sommes pas encore là, il est de plus en plus probable que des éléments extérieurs « naturels » (liés au « bruit » ambiant, et non à une faute de conception, de fabrication, à des problèmes d'alimentation ou à un environnement agressif) entraînent des dysfonctionnements du circuit. C'est le cas du rayonnement de neutrons, déjà bien connu dans le milieu spatial, qui peut entraîner dans les technologies futures des fautes dites « soft », de type Single Event Upset (SEU). La fiabilité des calculateurs aurait alors tendance à diminuer.

Un autre problème de fiabilité vient de l'introduction rapide des nouvelles technologies. Ce point a d'ailleurs été ajouté comme difficulté à surmonter dans la révision de la « road map » de 1998 [SIA1998].

Pour ces deux raisons, les problèmes de fiabilité des structures deviennent de plus en plus critiques.

3.3 Evolutions technologiques et implications architecturales

L'évolution technologique prévoit, à moyen terme, des circuits de plusieurs dizaines de millions de transistors sur une seule puce. L'abondance de transistors ouvre des possibilités architecturales inégalées jusqu'à présent. Elle pose également un problème de remise en cause des architectures actuelles et des méthodes de réalisation des circuits intégrés. Ainsi, la communauté de la microélectronique s'accorde à dire qu'avec une telle quantité de transistors, il est impensable de concevoir des circuits à plat, d'où la nécessité d'une hiérarchisation de l'architecture.

Par ailleurs, il semble impossible de concevoir l'intégralité d'un circuit de 10 millions de transistors en raison du coût d'une telle conception. Afin de pallier cette difficulté, la notion de propriété industrielle ou « intellectual property » (IP) est apparue. Son but est de permettre la réutilisation des fonctions de blocs provenant d'anciens circuits ou de fabricants qui proposent leur savoir-faire. Ce peut être, par exemple, des cœurs de processeurs autour desquels de nouvelles fonctions viendront se greffer. La compatibilité de ces blocs pose cependant un problème et fait actuellement l'objet de tentatives de normalisation. Les IP peuvent se présenter sous trois formes :

- une description comportementale (VHDL, Verilog,...) ;
- une description sous forme de bloc synthétisable avec ses contraintes d'optimisation ;
- un bloc « matériel », c'est à dire placé et routé selon une bibliothèque fondeur spécifique.

Le nombre important de transistors peut permettre d'augmenter la complexité du jeu d'instructions des processeurs. Ce point ne semble toutefois pas être une priorité, car les compilateurs ont déjà souvent du mal à suivre les innovations architecturales. La solution la plus couramment citée, en accord avec les deux contraintes de hiérarchisation et de réutilisation, consiste à augmenter le degré de parallélisme à l'intérieur même des circuits intégrés. Il est alors possible d'imaginer plusieurs types d'architectures à fort parallélisme [Burger and Goodman1997]. Parmi celles-ci, les calculateurs parallèles possèdent des avantages indiscutables. En effet, ces structures sont hiérarchisées et la répétition de leurs processeurs simplifie la conception. Ainsi, l'expérience accumulée au fil des années sur ces architectures fait de ces calculateurs parallèles des candidats intéressants pour les circuits intégrés futurs.

Dans une dizaine d'années, il sera peut-être possible d'intégrer un multiprocesseur entier sur une seule puce. Avant d'arriver à une telle extrémité, il est déjà possible d'intégrer une dizaine de processeurs de complexité moyenne dans un seul circuit intégré. En en associant plusieurs dans des MultiChip Module (MCM), un calculateur parallèle complet peut être contenu sur une seule carte électronique (figure 3). On obtient alors un calculateur parallèle très puissant et fortement intégré. Les avantages d'une telle intégration sont évidents, à commencer par l'amélioration des communications, une consommation et un encombrement réduits.

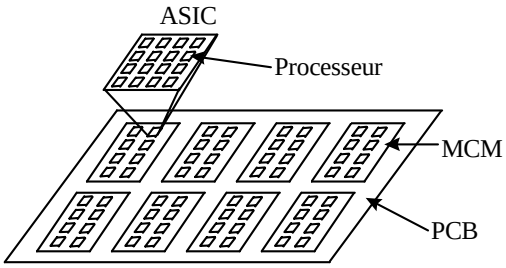


Figure 3 : réalisation physique d'un calculateur parallèle intégré

3.4 Conclusion

Les calculateurs parallèles possèdent des qualités qui en font de bons candidats pour les architectures des futurs circuits intégrés. Ceux-ci sont toutefois confrontés à des problèmes croissants en terme de fiabilité.

Il est intéressant de pouvoir quantifier la fiabilité actuelle de structures comme celle proposée (figure 3). Au paragraphe suivant, nous tentons d'apporter quelques chiffres afin de donner une idée de l'amélioration nécessaire de la fiabilité.

4 Fiabilité d'un composant et d'un calculateur parallèle

4.1 Introduction

Il est malheureusement très difficile d'obtenir des données stables sur la fiabilité des composants électroniques, et ceci pour deux raisons. La première est que ces chiffres ne sont en général pas accessibles à cause de la guerre commerciale que se livrent les grands acteurs du domaine. La deuxième raison est la difficulté intrinsèque pour obtenir de tels chiffres, difficulté due à l'évolution très rapide des filières électroniques. Une filière est bien définie au moment où elle devient obsolète ! De plus, il ne peut y avoir aucune généralité concernant la fiabilité. En effet, chaque filière de circuit intégrés, et plus que cela, chaque lot d'une filière a des caractéristiques qui peuvent être très différentes les unes des autres. Une solution est alors d'utiliser les bases de données prévisionnelles. Mais, en raison des difficultés citées, ces dernières restent imprécises.

Cette section s'articule ainsi : les concepts de fiabilité sont tout d'abord rappelés, puis les facteurs aggravant sont donnés ; enfin, la dernière partie de cette section traite de la limite des bases de données prévisionnelles.

4.2 Les concepts de mesure de la fiabilité

4.2.1 Taux de défaillance et fiabilité

Le concept fondamental utilisé pour le calcul de la fiabilité est celui de taux de défaillance. Le taux de défaillance Z(t) d'un composant ou d'un système est le nombre de défaillances prévues pendant un temps donné (en général par heure). Par ailleurs, la fiabilité R(t) d'un composant ou d'un système est la probabilité conditionnelle que le composant fonctionne correctement pendant un intervalle de temps $[t_o, t]$ sachant qu'il fonctionne correctement au temps t_o.

Parallèlement, on peut définir la non-fiabilité (ou probabilité que le composant ne fonctionne pas correctement durant l'intervalle $[t_o, t]$) d'un composant ou d'un système Q(t) = 1-R(t). dQ(t)/dt est alors appelée la densité de défaillance. On peut déduire de la définition du taux de défaillance Z(t) et de la fiabilité R(t) une relation pour t_o=0 :

$$Z(t) = \left(- \frac{dR(t) \,/\, dt}{R(t)} \right) \qquad (1)$$

Afin d'évaluer la qualité d'un système, un paramètre couramment utilisé est le temps moyen de défaillance, MTTF (Mean Time To Failure). Ce dernier représente une estimation du temps moyen avant la première défaillance. En terme de probabilité, c'est donc l'espérance E de la densité de défaillance dQ(t)/dt. On montre que le MTTF s'écrit :

$$MTTF = \int_0^\infty R(t)dt \qquad (2)$$

4.2.2 Hypothèses simplificatrices

Le taux de défaillance Z(t) est dépendant du temps. Toutefois, dans la littérature et pour les circuits électroniques, il est largement admis que ce taux peut être approché par une constante appelée λ pendant la période dite de vie utile (figure 4).

L'hypothèse de taux de défaillance constant suppose que le composant a été préalablement vieilli (burn-in) ou qu'il a passé la période dite de mortalité infantile, d'environ 3 mois. Cela suppose également que le composant n'atteint pas la région « catastrophique » de vieillissement accéléré. On estime en général que le système devient obsolète avant de pouvoir y arriver. Enfin, il faut que la fiabilité du composant corresponde bien à celle décrite par la courbe de la figure 4, ce qui peut être remis en question pour certaines technologies. Cette hypothèse simplificatrice sera néanmoins admise par la suite.

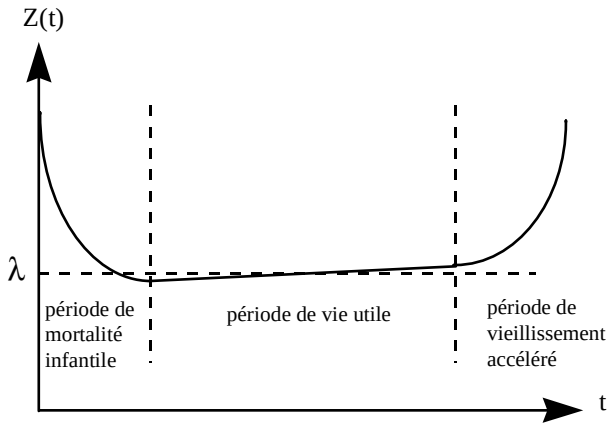


Figure 4 : courbe « en baignoire » représentant le taux de défaillance Z(t) en fonction du temps

Dans le cas d'un taux de défaillance constant, Z(t) = λ, la relation (1) devient une loi de défaillance exponentielle :

$$\boxed{R(t) = e^{-\lambda t}} \qquad (3)$$

Une fois la première hypothèse établie, la deuxième hypothèse couramment admise est qu'en l'absence de dispositif de tolérance aux fautes, le système est défaillant dès qu'un de ses composants l'est. Par conséquent, le taux de défaillance du système λ_S est égal à la somme des taux de défaillance élémentaires λ_i des différents composants :

$$\boxed{\lambda_S = \sum_i \lambda_i} \qquad (4)$$

4.2.3 Facteurs d'accélération

Un changement de paramètre interne ou externe au circuit intégré entraîne des modifications du taux de défaillance. Le rapport

taux de défaillance élémentaire le plus fort / taux de défaillance élémentaire le plus faible

est appelé facteur d'accélération (du vieillissement d'un circuit intégré). Les principaux facteurs d'accélération sont dus aux causes suivantes [ASTE1991] :

- la technologie

Les facteurs intervenant au niveau technologique portent sur des critères aussi différents que le type de transistor utilisé, sa taille, sa complexité ou encore le boîtier utilisé pour protéger le circuit. On tient également compte des classes de qualification des composants fournies par le constructeur et de facteurs empiriques comme l'ancienneté de la filière de fabrication.

- les contraintes d'environnement

Elles sont prises en compte par un coefficient multiplicateur empirique. Une dizaine d'environnements sont définis allant des conditions idéales de laboratoire à celles de missiles en phase de lancement, en passant par les conditions d'usine ou de satellite en orbite.

- les contraintes de fonctionnement

Ce sont les contraintes internes au composant, liées à la température de jonction des transistors, la puissance du circuit, sa tension.... Les variations de ces paramètres sont caractérisées par des courbes qui traduisent les phénomènes mis en jeu ou par des modèles mathématiques. Un des coefficients le plus important est le facteur d'accélération de la température noté π_t. Le modèle mathématique associé est la loi empirique d'Arrhenius :

$$\boxed{\lambda_2 = \lambda_1 \exp\left[(E_a / K)(1 / T_1 - 1 / T_2)\right]} \quad (5)$$

qui permet d'établir la relation entre deux taux de défaillance à différentes températures de jonction. Pour cette loi, K est la constante de Boltzmann (8,617.10^{-5} eV/°K) et E_a l'énergie d'activation associée au mécanisme de dégradation à l'origine des défauts. C'est en général l'énergie d'activation du défaut dominant qui est prise en compte.

Les valeurs usuelles de E_a sont :

- 0,5 eV lorsque l'on ne connaît pas le mécanisme de défaillance ;
- 0,3 eV s'il existe des défauts confirmés dont on ne connaît pas l'origine ;
- 0,7 eV si les expérimentations n'ont pas révélé de défauts ;
- une valeur empirique, lorsque cela est possible, pour les défauts localisés et identifiés (par exemple 0,5 à 1 eV pour des défauts de charges de surface).

Ainsi entre 40 et 70 °C, le facteur d'accélération est de 2,7 pour E_a=0,3 eV, 5,2 pour E_a=0,5 eV et atteint 10 pour E_a=0,7 eV !

D'autres lois sont définies à partir de la loi d'Arrhénius afin de mesurer l'influence d'autres paramètres extérieurs, comme l'humidité ou les surtensions [ASTE1991]. A titre de comparaison, pour apprécier l'importance des différentes contraintes :

- une augmentation de la tension d'alimentation de 5V à 8V donne un facteur d'accélération de 2,56 ;

- une augmentation du taux d'humidité relative de 50 % à 80 % donne un facteur d'accélération de 8 ;
- une augmentation de la température de jonction de 55 à 85°C pour E_a =0,7 eV donne un facteur d'accélération de 7,7.

L'action combinée de l'humidité et de la température peut être catastrophique pour la fiabilité du composant. En effet, les données de fiabilité de l'ASTE [ASTE1991] donne un facteur d'accélération de 1000 à 10000 pour le passage d'un état avec une température de jonction de 55 °C et un taux d'humidité de 50 % à un état avec une température de jonction à 85 °C et un taux d'humidité de 85 %.

Toutefois, la température sera un élément prédominant dans un environnement peu agressif, car il est possible de lutter efficacement contre l'augmentation d'humidité relative (boîtier étanche) alors qu'il est plus difficile de maîtriser une augmentation de la température.

4.2.4 Les limites des bases de données de fiabilité prévisionnelle

La fiabilité d'un composant dépend des énergies d'activation mises en jeu. Or la connaissance de ces énergies d'activation dépend des mesures. Le calcul d'un taux de défaillance est donc très délicat. Il ne peut reposer que sur une suite d'expériences réalisées en utilisant plusieurs fois un composant issu d'une même filière technologique. Par conséquent, des facteurs comme le type de technologie, leur maîtrise, les quantités produites ou la date de fabrication sont autant de critères difficiles à prendre en compte et qui jouent un rôle important dans la prévision.

Les problèmes sont encore plus importants lorsqu'il s'agit de qualifier un circuit spécifique (ASIC). Ces circuits, produits à un petit nombre d'exemplaires ne peuvent pas être qualifiés par des moyens conventionnels qui consistent à faire vieillir artificiellement un grand nombre de composants. En effet, cela nécessiterait plus de circuits pour les tests que pour l'application finale. On qualifie donc plutôt les « familles » d'ASIC [DoD1995], c'est à dire les ASIC réalisés avec la même technologie, conçus selon les mêmes règles de dessin et élaborés avec le même procédé de fabrication.

Les documents les plus utilisés pour obtenir des données prévisionnelles quantitatives de fiabilité relatives aux composants et aux systèmes sont le « Military Handbook » Mil-Hdbk-217 [DoD1995] et Bellcore de AT&T [AT&T1997]. Ces deux modèles sont à peu prêt équivalents, bien que Bellcore soit un peu plus complet. Toutefois, le premier de ces deux documents est actuellement très controversé. Notamment l'AMSAA (U.S. Army Matériel Systems Analysis Activity) montre par des tests directs que les prédictions du Mil-Hdbk-217 peuvent être fausses jusqu'à un facteur 20 [AMSAA1995]. Les reproches principaux faits au Mil-Hdbk-217 portent sur ses bases techniques et technologiques « insondables » et l'absence d'évaluation de l'incertitude concernant les prédictions. Pour toutes ces raisons, un facteur de 1 à 2 en précision de prévision est considéré comme un résultat correct !

Le calcul de fiabilité qui est proposé au paragraphe suivant ne repose pas sur des bases scientifiques solides. Il nous a néanmoins paru nécessaire de l'intégrer à ce mémoire pour donner un ordre de grandeur de la fiabilité d'un calculateur parallèle. Il nous permet également d'illustrer les effets des facteurs d'accélération.

4.3 Calcul de la fiabilité d'un calculateur parallèle type

Afin de donner quelques chiffres concrets de la fiabilité d'un ASIC, nous supposerons que la fiabilité des composants dans une technologie avancée (pas encore stabilisée) n'a guère évoluée depuis le début des années 80. En effet, les avancées technologiques ont permis d'améliorer de façon considérable les outils de fabrication, mais les technologies de plus en plus fines sont de plus en plus susceptibles de produire des fautes. Nous faisons ici l'hypothèse que le rapport entre ces deux phénomènes est resté à peu prêt constant.

Pour notre calcul, nous prenons comme référence :

- le document de données de fiabilités du CNET de 1983 [CNET1983] ;
- un calculateur de 1024 processeurs répartis en 64 ASIC de 16 processeurs (repris tout au long de ce mémoire).

Nous ajoutons ici l'hypothèse que le calculateur ne possède pas de mécanisme de tolérance aux fautes. Le système est donc fautif dès qu'un de ses composants l'est. Par ailleurs, nous considérons qu 'un ASIC comporte 10000 portes logiques, ce qui constituait la référence en terme d'évolution dans le document du CNET.

Enfin, nous supposons que la cause dominante d'accélération du vieillissement des ASIC est une augmentation de leur température et que l'énergie d'activation de ce phénomène est Ea = 0,7 eV.

Dans ces conditions, le taux de défaillance d'un ASIC à une température de jonction des transistors de 40 °C est de 8550.10^{-9}/h, soit un MTTF de 13,5 ans. Ce chiffre est à mettre en relation avec l'objectif de 5 à 25 ans donné par la « road map » du SIA en 1998 [SIA1998]. A une température de jonction de 70 °C, le taux de défaillance d'un ASIC passe à 26500.10^{-9}/h, soit un MTTF de 4,5 ans.

Pour 64 ASIC, le MTTF est alors de :

- $\frac{13.5}{64} = 2{,}5$ mois à 40 °C ;
- $\frac{4.5}{64} = 26$ jours à 70 °C.

Les processeurs peuvent être constitués d'une partie importante de mémoire vives de type Random Access Memory (RAM). Il est donc intéressant d'effectuer le calcul précédent en tenant compte de cette caractéristique. Selon le document du CNET, les éléments de mémorisation sont environ 2 fois plus fiables que les circuits intégrés en logique aléatoire. Une structure dont les processeurs seraient formés de mémoire à 50 % serait donc plus fiable, à savoir :

Pour 64 ASIC, un MTTF de

- $\frac{13.5 + 29.6}{2 * 64} \cong 4$ mois à 40 °C ;
- $\frac{4.5 + 10}{2 * 64} \cong 1.3$ mois à 70 °C.

4.4 Conclusion

Les données de fiabilité présentées ci-dessus sont évidemment à considérer avec la plus grande prudence et ne sont que purement indicatives. Toutefois, elles montrent que le groupement d'un grand nombre de composants augmente fortement la probabilité de défaillance du système et que des facteurs comme la température de jonction des transistors influent beaucoup sur la fiabilité des calculateurs intégrés. La section suivante dresse un inventaire des solutions permettant d'augmenter la fiabilité. Nous y choisirons l'approche la plus pertinente dans le cas des calculateurs parallèles intégrés de type SIMD.

5 Les solutions pour améliorer la fiabilité

5.1 Introduction

Comme il est rappelé dans l'Annexe 1, il existe deux types de méthodes permettant d'améliorer la sûreté de fonctionnement du système : éviter ou tolérer les fautes.

Par construction, il est possible d'éviter qu'une faute ne se produise. La fiabilité est améliorée par la filière technologique utilisée. Différentes solutions existent :

- utilisation d'une technologie stabilisée ;
- utilisation de technologie durcie qui permet de lutter efficacement contre les effets des radiations (par exemple la technologie DMILL développée au CEA [RD291997]) ;
- utilisation des technologies de Silicium sur Isolant (SOI) [Bensahel and Bomchil1989, Faynot1998] : « caissons » d'isolation pour les transistors qui les rendent plus résistants aux champs électromagnétiques ;
- jouer sur les règles de dessin du circuit. Certaines parties d'un circuit peuvent ainsi être conçues de façon à minimiser la probabilité d'apparition de phénomènes liés, par exemple, à l'électromagnétisme.

Ces techniques impliquent que l'amélioration de la fiabilité soit liée à la technologie et aux règles de dessin. Il est toutefois souhaitable que les moyens utilisés pour augmenter la fiabilité ne dépendent pas uniquement de la construction matérielle. De cette façon, il est possible de conserver l'indépendance du circuit vis à vis de la technologie et d'assurer une fiabilité montante à plusieurs niveaux.

La solution de tolérance aux fautes s'impose alors. Elle suppose qu'un défaut du système entraîne un dysfonctionnement de la structure ou faute. Le but de la tolérance aux fautes est alors d'éviter que cette faute ne se propage pour devenir une erreur, puis une panne.

5.2 Tolérance aux fautes

5.2.1 Introduction

Un point commun à toutes les techniques de tolérance aux fautes est l'utilisation d'une redondance. Elle peut être matérielle (certains modules matériels sont dupliqués), de temps (certaines parties d'un

programme sont exécutées plusieurs fois ou l'algorithme est effectué de plusieurs façons différentes), d'information (le circuit ou le programme possède une redondance d'information) ou un mélange de ces trois solutions. L'éventail des méthodes existantes peut être dressé :

- les méthodes logicielles qui utilisent une redondance temporelle (N-version programming) ou d'information ;
- les méthodes utilisant une redondance d'information matérielle ;
- les méthodes de redondance aux fautes temporelle ;
- les méthodes de routage tolérant aux fautes qui utilisent une redondance matérielle ou de temps (on parle alors de mode dégradé pour le fonctionnement du système) ;
- les méthodes de tolérance aux fautes avec redondance de matériel et de façon statique ;
- et enfin, les méthodes de tolérance aux fautes avec redondance de matériel et de façon dynamique.

Il est à noter que 3 de ces solutions sont dédiées aux calculateurs parallèles. Les avantages et inconvénients de ces 6 méthodes sont maintenant discutés.

5.2.2 Les méthodes logicielles

Ce sont des solutions pour rendre un programme ou une partie de programme tolérant à certaines fautes matérielles (à ne pas confondre avec la tolérance aux fautes des logiciels). Il existe un nombre important de possibilités d'insérer de la tolérance aux fautes par logiciel. Nous pouvons citer, à titre d'exemple :

- les méthodes à redondance temporelle qui consistent à répéter des instructions, des segments de programme ou des programmes entiers. Celles-ci permettent la détection et la tolérance des fautes momentanées du système mais ne permettent pas de détecter des fautes permanentes de la structure.
- les méthodes qui consistent à ajouter de l'information, par exemple en exécutant chaque fonction f et sa fonction self-dual $\overline{f}$ (i.e. les fonctions f et $\overline{f}$ telles que $\overline{f}(x_1, x_2, \ldots, x_n) = f(\overline{x_1}, \overline{x_2}, \ldots, \overline{x_n})$).

Certaines de ces techniques sont regroupées sous le nom plus générique de Algorithm Based Fault-Tolerance (ABFT) . Elles peuvent se décliner sous plusieurs formes, et des exemples peuvent être trouvés dans [Chowdhury and Banerjee1996, Oh1995, Vaidya1998].

Avantage

La tolérance aux fautes est obtenue sans modification de la structure matérielle.

Inconvénients

La modification des programmes est souvent très lourde à mettre en œuvre et la perte en performance induite par ces solutions est difficilement acceptable pour des accélérateurs de calcul.

5.2.3 Redondance d'informations obtenue par matériel

Cette méthode de tolérance aux fautes consiste à coder les données de façon matérielle de manière à ce qu'elles fournissent un supplément d'information. Celui-ci permet alors de détecter et/ou de corriger les fautes. C'est une méthode très utilisée en raison de sa facilité d'utilisation pour certains composants, telles les mémoires. Le code le plus connu et le plus couramment utilisé est le code de parité qui permet de détecter la présence d'une faute dans un mot. Ce code ne permet pas de corriger les fautes, mais seulement de les détecter. Il doit donc être couplé à d'autres méthodes pour assurer la tolérance aux fautes.

Il est alors possible d'utiliser d'autres codes qui permettent la correction des fautes en plus de les détecter. Parmi ceux-ci, nous pouvons citer les codes de Berger, les codes de Hamming utilisés pour le test des mémoires, les codes de parité 2-D et les codes arithmétiques [Fujiwara and Matsuoka1987, Khakbaz and McCluskey1984, Noraz1989].

Avantages

Ces méthodes sont transparentes vis-à-vis de la couche logicielle et donc de l'utilisateur. Par ailleurs, la diminution de performance est beaucoup moins sensible que pour les techniques logicielles. De plus, une faute est détectée très rapidement. Pour ces trois raisons, ces techniques sont développées dans la partie traitant du test afin de couvrir les fautes transitoires.

Inconvénients

Tout d'abord, le coût matériel de ces méthodes peut être important, notamment pour la correction des fautes et les pertes en performance induites peuvent être tout de même assez conséquentes, de l'ordre de 20 à 30 % pour un processeur simple (voir chapitre 3). Par ailleurs, cette approche nécessite d'agir directement sur la conception structurelle des blocs logiques (voir chapitre 2). Cela pose le problème de l'insertion de ces techniques au niveau de la conception, même si des efforts ont été faits pour fournir des outils à plus haut niveau [Duarte1997, Nicolaidis1993, Nicolaidis1997]. Enfin, ces techniques excluent de pouvoir utiliser les blocs matériels optimisés des librairies de synthèse (comme des blocs multiplieur).

5.2.4 Redondance temporelle obtenue par matériel

Cette technique concerne plus particulièrement les calculateurs parallèles. La méthode de redondance temporelle consiste à répartir les tâches d'un processeur fautif sur un ou plusieurs processeurs en état de fonctionnement [Negrini and Stefanelli1986]. On parle alors de mode dégradé pour le fonctionnement du système, puisque la structure est « ralentie » par les processeurs fautifs. Afin de répartir les tâches, le réseau est reconfiguré de façon à ce que les données devant aboutir au processeur fautif soient distribuées vers des processeurs corrects.

Avantages

Un faible coût matériel apporte une amélioration très sensible de la fiabilité de la structure. En outre, la méthode est transparente vis à vis du logiciel.

Inconvénients

L'inconvénient de ce type de tolérance aux fautes se traduit par une perte de performance graduelle. Cette solution ne peut donc pas être retenue pour des applications temps réel comme le traitement d'image ou des applications gourmandes en temps de calcul. Or, celles-ci constituent les principales applications des calculateurs intégrés.

5.2.5 Routage tolérant aux fautes

Une solution astucieuse de tolérance aux fautes dédiée aux calculateurs parallèles est de rendre tolérant aux fautes le routage des données qui transitent entre les processeurs. La méthode s'applique aux calculateurs dont les communications se font par passage de message. Elle consiste à modifier l'algorithme de routage, de façon à faire parvenir les données envoyées vers un processeur fautif à un processeur remplaçant [Boppana and Chalasani1995, Leighton and Maggs1992, Oh1995, Sui and Wang1997, Tzeng1993].

Avantages

La tolérance aux fautes est simple à mettre en place et n'occasionne qu'un faible coût matériel quand les processeurs sont de taille importante. Lorsque des processeurs supplémentaires sont utilisés, la méthode est également transparente vis à vis du logiciel.

Inconvénients

En général, les calculateurs SIMD n'utilisent pas de routage par passage de message en raison de leur synchronisme. D'autre part, le routage tolérant aux fautes est lui-même mis en défaut lorsqu'un module de routage ne remplit plus son office, ces modules sont donc dits critiques. Enfin, cette méthode entraîne des pertes en performance importantes lorsque les communications sont régulières (les processeurs émettent leurs données dans la même direction). En effet, une donnée peut avoir à parcourir un chemin important pour aller d'un processeur à son voisin logique. Le routage tolérant aux fautes n'est donc pas retenu dans le cadre de cette étude.

5.2.6 Tolérance aux fautes de façon matérielle et statique

Cette méthode utilise une redondance matérielle de façon à « masquer » la faute : celle-ci n'est pas forcément détectée, mais le résultat en sortie de la structure reste correct. Ces architectures sont également appelées structures de vote. La réplication de trois modules identiques (Triple Modular Redundancy ou TMR [Kim and Shin1996, Wakerly1976]) en est un exemple classique. Les sorties de ces modules sont alors comparées par le moyen d'un voteur qui constitue la partie critique du système. Il est alors possible d'utiliser plusieurs voteurs pour diminuer le risque ou de rendre ce voteur très fiable [Kim and Shin1996]. La première solution est souvent préférée : l'Airbus A320, par exemple, utilise pour les commandes un système de 5 calculateurs dont les résultats sont comparés par 3 voteurs.

Avantages

La méthode est transparente vis-à-vis du logiciel et les pertes en performance sont minimes.

Inconvénient

Le surplus matériel est très important (plus de 200 %). Il ne peut être acceptable que pour des systèmes critiques de type avionique ou contrôle de trafic.

5.2.7 Tolérance aux fautes matérielles dynamique

L'approche de la tolérance aux fautes dynamique concerne uniquement les calculateurs parallèles. La redondance est de type matériel, mais elle est limitée à quelques processeurs supplémentaires « dormants » qui n'interviennent pas dans la fonctionnalité du calculateur. Lorsqu'un processeur de la structure primaire est considéré comme fautif, il est alors remplacé par un des éléments « dormants ».

Avantages

Cette approche est indépendante vis-à-vis du logiciel, la diminution des performances est faible et le coût matériel limité.

Inconvénients

Il est difficile de bien gérer le compromis entre le surplus de matériel et le besoin en fiabilité. La méthode est souvent complexe à mettre en oeuvre.

5.3 Conclusion

Dans cette section, nous avons présenté 6 méthodes de tolérance aux fautes. Celles-ci ont été développées avec des objectifs différents en terme de caractéristiques. Les avantages et inconvénients de chaque méthode sont rappelés dans le tableau 1. Les critères pris en compte sont les suivants :

- l'indépendance vis-à-vis du logiciel (1)
- le coût matériel (2)
- le coût en performance (3)
- la tolérance des fautes permanentes (4)
- la tolérance des fautes transitoires (5)
- la facilité de mise en œuvre de la méthode (6)

En raison de nos contraintes particulières, les conclusions tirées de ce tableau sont les suivantes :

Les méthodes logicielles et de redondance temporelle sont écartées en raison des pertes en performance qu'elles engendrent. Le routage tolérant aux fautes est écarté pour cette même raison, bien qu'il reste un bon candidat à la tolérance aux fautes de certains multiprocesseurs. Enfin, la redondance matérielle statique est rejetée à cause de son trop grand coût matériel.

Nous choisissons donc d'utiliser une redondance d'information afin de traiter le cas des fautes transitoires lorsque cela est nécessaire, et une redondance matérielle dynamique pour traiter le cas des fautes permanentes et améliorer la durée de vie du système. Nous présentons en conclusion de ce chapitre la méthodologie générale d'amélioration de la fiabilité basée sur ces deux méthodes.

	1	2	3	4	5	6
logiciel	non	****	*	oui	oui	*
information (codage)	oui	**	***	non	oui	* à *** selon le code
temporel	oui	***	**	oui	non	***
routage	oui	***	** pour SIMD **** pour MIMD	oui	non	***
mat. statique	oui	*	****	oui	oui	***
mat. dynamique	oui	***	*** à ****	oui	non	***

Tableau 1 : récapitulatifs des avantages et inconvénients des différentes méthodes de tolérance aux fautes (* = inconvénient fort, **** = avantage fort)

6 Conclusion : Méthodologie générale

La méthodologie générale est liée à la tolérance aux fautes dite « structurelle ». Cette dernière permet de conserver la structure (ou la topologie) de l'architecture en cas de faute. Elle est décrite par les points suivants :

- détection de la faute ;
- localisation de la faute ;
- vérification de la nature non transitoire de la faute ;
- recouvrement des bonnes données de l'élément fautif ;
- reconfiguration du système, en remplaçant l'élément fautif par un élément « dormant ».

Ces 5 points constituent la trame suivie tout au long de ce mémoire. Nous séparerons la suite de cette présentation en deux parties assez distinctes : le test, notre réponse aux deux premiers points, et la tolérance aux fautes elle-même, traitée sous l'angle de la reconfiguration du système.

Le test est le premier problème abordé. Il doit répondre aux deux questions de détection et de localisation d'une faute transitoire ou permanente en fonctionnement normal. De plus, pour des raisons d'économie de moyens et de cohérence générale, ce test doit également répondre aux questions de test de fabrication. Ceci sera détaillé dans les chapitres 2 et 3.

La vérification de la nature transitoire ou non de la faute est réalisée grâce à une relance de la procédure de test : une période, dite « de faute transitoire », est utilisé à cet effet. Elle correspond à un certain nombre de boucles de test. Si le circuit reste fautif durant cette période, la faute est considérée

comme permanente. On doit alors faire appel à des méthodes de reconfiguration du système. Sinon, la faute est transitoire et la reconfiguration n'est pas effectuée. En cas de faute considérée comme permanente le processeur fautif doit continuer à être testé, ceci afin qu'un circuit déclaré fautif et supprimé logiquement de la structure puisse être à nouveau intégré dans celle-ci si son fonctionnement redevient correct.

L'étape suivante concerne le recouvrement de contexte. Cette opération peut être effectuée à ce niveau ou après la reconfiguration du système. Elle consiste à récupérer un état correct du système à partir duquel l'application pourra être relancée. Cette étape est inutile pour certaines applications, dont la plupart de celles dites en temps réel. On peut citer pour exemple le traitement d'images à la fréquence vidéo et, de manière plus générale, toute application utilisant des données rafraîchies à cadence rapide. Lorsque cette étape est utile, plusieurs solutions sont possibles. Il est ainsi envisageable de partager les données de chaque processeur sur ses voisins. Il est aussi possible d'utiliser une grande mémoire (dite mémoire de masse) dans laquelle on sauvegarde périodiquement toutes les données du système. Pour notre part, nous supposons que c'est cette dernière solution qui est appliquée, et nous verrons comment la rendre viable pour des calculateurs parallèles.

La reconfiguration sera le thème central des chapitres 4, 5 et 6. Les différentes solutions existantes sont tout d'abord évoquées dans le quatrième chapitre, puis leurs limites sont discutées. Dans le cinquième chapitre, nous proposons une solution afin de passer outre ces limitations. Le sixième chapitre est axé sur l'architecture de réseaux tolérants aux fautes. Il sert également à calculer l'amélioration de la fiabilité qu'il est possible d'obtenir par la méthode proposée.

Pour des raisons de lisibilité du mémoire, nous avons choisi de séparer nettement les chapitres concernant le test de ceux traitant de la reconfiguration. Pour cette raison, ces 2 parties ont été conçues de façon à être le plus possible indépendantes et autosuffisantes, et certaines sections peuvent être légèrement redondantes entre elles ou avec cette introduction.

Chapitre 2 : le test des circuits intégrés

1 Introduction

A notre connaissance, aucun calculateur SIMD ou multi-SIMD n'a fait l'objet d'une méthodologie de test incluant des méthodes de test précises adaptées à la fois à la fabrication et au test lors du fonctionnement. Par exemple, la CM-2 de Thinking Machine Corporation (TMC) utilise un simple programme de test fonctionnel de quelques minutes tous les jours et la machine ELSA développée au laboratoire CSI à Grenoble utilise un test de fabrication basé sur un microscope électronique à balayage (MEB).

Par conséquent, le but de ce chapitre est de présenter le problème du test des circuits intégrés afin de positionner notre approche. Pour cela, il est nécessaire de définir précisément la notion de test qui est liée à celle de validation (d'un concept logiciel ou d'une structure physique). Ainsi, nous verrons successivement :

- les différentes phases de test ;

Cette section s'attache à décrire les tests nécessaires aussi bien durant la conception d'un système et de ses composants matériels qu'en fin de fabrication ou pendant le fonctionnement d'un circuit intégré.

- les modèles de fautes ;

Une méthode de test, quel que soit ce test, consiste à mettre en relief certains défauts d'un système (logiciel, matériel ou mixte) pouvant entraîner un dysfonctionnement. Pour ce faire, on se munit de modèles de ces défauts que l'on cherche à observer. Ces modèles sont conçus par rapport à la manifestation des défauts, qui est appelée une faute. Ils influent fortement sur la composition des procédures de test et sur leur qualité.

- les méthodes de test.

A la suite des deux premières sections, une troisième section fait un tour d'horizon rapide des différentes possibilités de test. Ces méthodes sont alors comparées afin d'expliquer l'approche qui sera suivie concernant le test d'un calculateur parallèle SIMD ou multi-SIMD.

2 Les différentes phases de test

2.1 Introduction

La figure 1 présente les phases simplifiées de conception et de validation d'un système. A partir de spécifications qui sont données par une application ou un ensemble d'applications, on réalise une répartition des tâches entre le matériel et le logiciel et on en définit l'interface. L'étape suivante consiste à traduire chaque partie dans le langage adéquat, par exemple :

- Specification and Description Langage (SDL) pour le comportement d'un système opératoire (OS) qui forme l'interface ;
- C, C++, Java... pour l'écriture de la partie logicielle ;
- VHSIC Hardware Description Langage (VHDL) pour la modélisation de la partie matérielle.

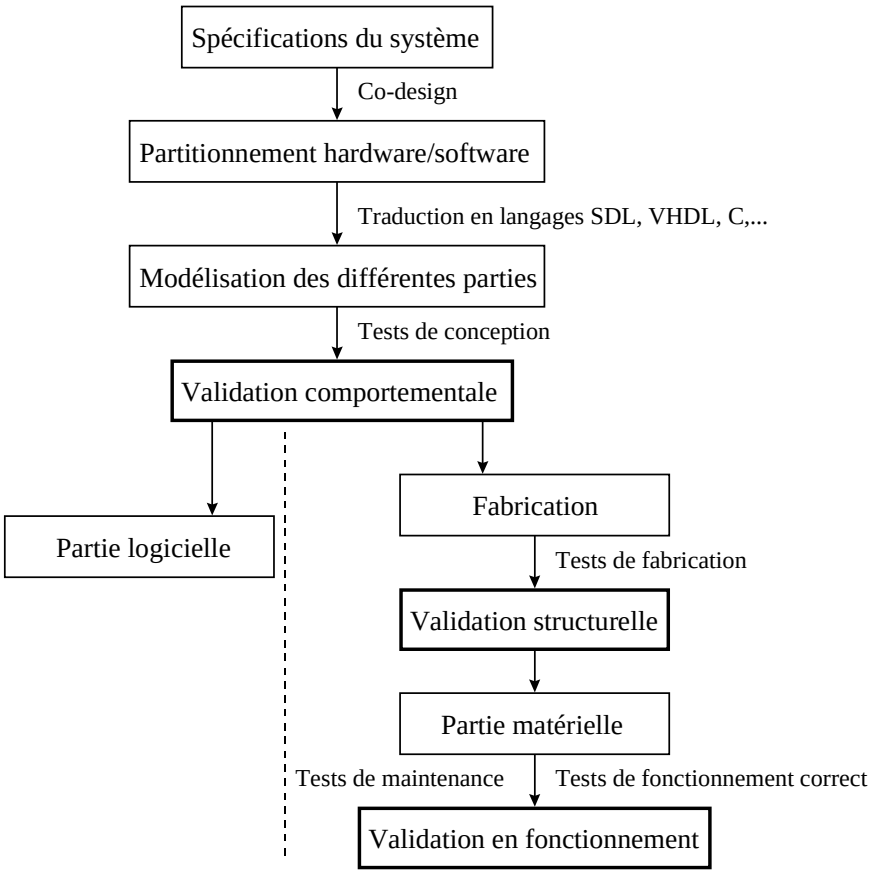


Figure 1 : les différentes phases de conception et de validation d'un système

Les modélisations de chacune des parties du système se font en plusieurs étapes. Lorsque la partie matérielle mène à la réalisation d'un circuit intégré spécifique (ASIC), la méthode générale consiste à partir d'un modèle VHDL représentant le comportement du circuit pour aboutir à une description de la couche physique (layout). Des descriptions intermédiaires sont également utilisées. Parmi celles-ci, nous pouvons citer le niveau de transfert de registres (Register Transfert Level RTL).

A la suite de chacune des étapes menant à la réalisation d'un circuit intégré ou d'un logiciel, il faut valider les descriptions successives, soit entre elles, soit par rapport aux spécifications de départ. C'est le but de la première phase de validation. On parle alors de validation de la conception pour le matériel. Cette première phase est très différente pour les parties matérielle et logicielle car la difficulté et les buts ne sont pas les mêmes : pour le matériel, on demandera à être sûr à 100 % du modèle, car une erreur de conception coûte très cher. Une telle demande est très difficilement concevable du côté du logiciel et on cherchera plutôt à minimiser les erreurs.

Les phases de test sont alors terminées en ce qui concerne le logiciel, même si en pratique on continue à corriger les bugs pendant le fonctionnement du programme. Au contraire, les phases de test de la partie matérielle ne sont pas terminées. En effet, il faut encore valider la structure physique obtenue après fabrication, car cette structure peut être imparfaite. C'est le rôle de la deuxième phase de test. Enfin, en raison des effets conjugués du vieillissement des composants et des agressions extérieures (hausse de température, variations de l'alimentation,...), la structure physique demande à être validée au cours du fonctionnement du système par des tests de maintenance et/ou des tests « en-ligne ». C'est la troisième phase de test.

Les tests logiciel et matériel sont donc très différents : dans le premier cas, il s'agit de vérifier ou de prouver que les descriptions successives correspondent bien aux spécifications ; dans le second cas, le but est de contrôler que la réalisation physique est correcte et ne dévie pas des spécifications au cours du temps. Le but fixé dans ce mémoire est d'assurer une meilleure fiabilité de la partie matérielle du système. Nous nous focaliserons donc sur le test des structures matérielles qui est maintenant décrit à travers ses 3 phases.

2.2 Validation de la conception

La conception matérielle repose sur des flots de conception dont un exemple est représenté figure 2. L'idée générale d'un tel flot est de partir d'un niveau de description élevé déduit des spécifications générales pour aboutir, par raffinements successifs, à une description assez proche du matériel. Cette description est alors utilisée par les compilateurs de silicium afin d'obtenir le dessin de la couche physique du circuit intégré.

Il est nécessaire de valider une description par rapport à une autre ou par rapport aux spécifications. On s'assure ainsi que les différents modèles d'une même structure sont équivalents et correspondent bien aux spécifications demandées.

Plusieurs solutions de validation de la conception sont possibles, selon les niveaux de description. Nous pouvons citer, sans être exhaustifs :

- La relecture des programmes (revue de codes) ;
- La preuve et vérification formelle ;
- La simulation ;
- L'émulation ;
- L'utilisation d'outils permettant de remonter la chaîne de conception (par exemple le VHDL VITAL).

La première solution est la relecture des programmes ou revue de codes. Une personne extérieure au projet (au mieux) relit le code. La simplicité d'une telle méthode ne doit pourtant pas cacher son extrême efficacité...

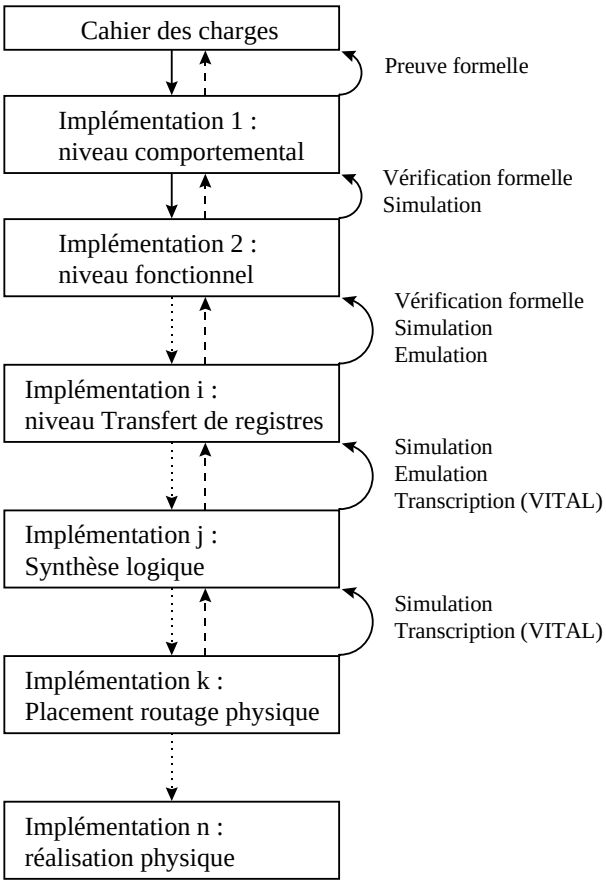


Figure 2 : chaîne de conception descendante et les différentes méthodes de test de conception applicables

La preuve et vérification formelle consistent à prouver des propriétés des différentes descriptions ou à prouver l'équivalence entre ces propriétés. Une telle approche assure en théorie 100 % de fiabilité des résultats, mais est malheureusement limitée en raison de la complexité de sa mise en œuvre.

La simulation est une des solutions les plus utilisées. On définit pour le système des « vecteurs de test » qui doivent représenter le fonctionnement du système. On traduit alors au fur et à mesure ces vecteurs de test selon le niveau de description utilisé. L'idéal est alors de pouvoir les utiliser pour valider la nouvelle description. En raison de la complexité croissante des systèmes, cette solution devient de plus en plus lourde à mettre en œuvre et nécessite souvent des temps de calcul très importants.

Pour contrer cela, on peut alors utiliser un émulateur. L'émulation consiste à simuler le système sur un support matériel, comme par exemple une « mer » de circuits programmables (FPGA). Les avantages sont nombreux, puisque le test peut se faire à fréquence importante (voire à la fréquence nominale du circuit) et on peut récupérer un grand nombre de variables internes du circuit.

Enfin, il existe des outils permettant de remonter le flot de conception. C'est le cas, par exemple, d'outils faisant la traduction de modèles composés de blocs décrits dans une bibliothèque physique (appelée bibliothèque fondeur) en un langage VHDL normalisé appelé VITAL. Il est alors possible de simuler ce VHDL et ainsi de valider le niveau de conception formé par les blocs physiques.

La validation de la conception fait donc intervenir une approche purement fonctionnelle du système physique sans tenir compte de la structure même du circuit (l'émulation, par exemple, utilise une structure complètement différente de celle qui est ensuite conçue). Cette validation, suffisante pour une structure logicielle, n'est qu'une étape de la validation de la structure matérielle. Afin d'étayer notre propos, imaginons qu'une instruction d'un système consiste à lire une donnée en mémoire. Le test de conception devra vérifier qu'à l'application de cette instruction, la donnée en mémoire est bien lue. C'est insuffisant pour le test matériel qui demande de vérifier non seulement que la donnée est lue, mais également que la donnée lue est correcte car un défaut sur la mémoire peut avoir altéré cette donnée. C'est à ce dernier niveau que les problèmes de fiabilité apparaissent. Par conséquent, nous ne traiterons pas la validation de la conception dans ce mémoire.

2.3 Validation de la fabrication

La validation de la fabrication est la vérification que la structure même du circuit a été réalisée correctement, selon les spécifications. Il s'agit donc de valider la construction physique de la structure. Ce test varie selon la technologie employée, la filière technologique et la réalisation physique. Cette dernière dépend de facteurs comme la qualité des matériaux utilisés, les techniques employées pour chaque phase de réalisation (épitaxie, réalisation des masques, attaques chimiques...) et leur enchaînement.... Heureusement, les défauts apparaissant sur les circuits ont des conséquences similaires pour toutes les filières technologiques d'une même famille, ce qui permet de classer les défauts et leurs conséquences. C'est l'objet du paragraphe traitant des modèles de fautes.

Les méthodes pour le test de fabrication sont données dans une section ultérieure. Il est à noter qu'elles peuvent en général être reprise pour le test de maintenance.

2.4 Vérification en cours de fonctionnement

Au cours du fonctionnement d'un système, l'environnement extérieur ainsi que le vieillissement naturel des composants peuvent entraîner des dysfonctionnements. Ceux-ci peuvent être soit temporaires, soit permanents. Il est donc nécessaire de vérifier que la structure physique répond toujours aux spécifications de départ. Les solutions de test pour ces dysfonctionnements sont présentées dans une section ultérieure.

2.5 Conclusion

Pour obtenir un test efficace tout à fait complet de la structure matérielle, il est nécessaire de disposer de trois sortes de test traitant des problèmes : de conception, de fabrication et de maintenance, et de dysfonctionnements apparaissant lors du fonctionnement du circuit.

Pour traiter des tests de la réalisation physique du système, il est nécessaire de se poser la question des grandes classes de dysfonctionnements affectant les circuits intégrés. Ce problème complexe est maintenant abordé par l'intermédiaire de la notion de faute.

3 Les fautes

3.1 Introduction

Définition : une faute est la manifestation d'un défaut physique du circuit.

Le problème à la base d'une des principales difficultés pour la réalisation de tests efficaces est le problème de la définition et de la caractérisation des « classes » ou « modèles » de fautes. Le paragraphe qui suit vise à donner ces différents modèles. La limitation de ceux-ci est ensuite discutée. Enfin, nous abordons les problèmes de contrôlabilité et d'observabilité des fautes avant de conclure cette section.

3.2 Fautes et défauts physiques

La cause des défauts est complexe et dépend de la technologie mise en œuvre, des phénomènes physiques qui s'y rattachent, de la « chaîne » de fabrication ainsi que de l'usage dont on fait du circuit : par exemple, certains défauts n'apparaîtront qu'à partir d'une certaine température de fonctionnement. Les défauts des circuits peuvent provenir de phénomènes physiques naturels comme l'électromigration ou les décharges électrostatiques ou de défauts de fabrication, notamment au niveau des masques utilisés ou des poussières se déposant sur le substrat. Ce dernier point est le moins facilement maîtrisable, car les renseignements sur ce type de défauts et surtout leur fréquence doivent provenir des fondeurs qui, on le comprend facilement, laissent difficilement filtrer ces informations. De plus, ces fautes dépendent de la maturité de la technologie qui est extrêmement difficile à évaluer.

Dans le domaine des semi-conducteurs, une liste non-exhaustive des défauts peut être dressée, chaque technologie ajoutant de nouvelles possibilités de défauts ou supprimant certains défauts. Nous pouvons citer, à titre d'exemples, la contamination ionique, les problèmes de charge (inversions, accumulation, pertes), la polarisation, les défauts d'oxyde, les porteurs chauds, le claquage diélectrique, les interactions entre matériaux comme la migration métallique, les problèmes de corrosion, de fissures, d'électromigration et les défauts de silicium et de masque [ASTE1991]. Nous verrons toutefois qu'il est possible de limiter le nombre de modèles de fautes en les classant en catégorie selon leurs conséquences au niveau électrique, fonctionnel ou comportemental.

Une caractérisation importante des fautes est la durée de leur action sur le système. On distingue les fautes ayant un effet temporaire sur le circuit et les fautes à caractère permanent. Les premières n'affectent les résultats que pendant un temps fini relativement court et sont appelées fautes transitoires. Les secondes affectent définitivement le fonctionnement du circuit. Les fautes transitoires peuvent être dues à des éléments extérieurs du circuit comme une légère surtension de l'alimentation électrique. Les dernières technologies peuvent également entraîner l’apparition d’événements isolés dus aux ions lourds. Les fautes transitoires, contrairement aux permanentes ne demandent pas à réparer le circuit.

3.3 Quels modèles de fautes ?

3.3.1 Introduction

Lorsqu'on veut tester un composant, une solution qui vient aussitôt à l'esprit est de mettre à l'entrée du circuit toutes les combinaisons possibles puis de vérifier en sortie de celui-ci que les résultats sont corrects. Cela se heurte à plusieurs problèmes.

Le premier est que l'approche exhaustive n'est bonne que pour des blocs combinatoires. Les blocs séquentiels ne sont pas testés entièrement de cette façon. Pour la logique séquentielle, une approche par suite de vecteurs de test est totalement impossible à mettre en place car le nombre de vecteurs de test qu'il faudrait exploserait très rapidement.

Un deuxième problème est que certains défauts entraînent des comportements séquentiels de blocs combinatoires. Enfin, il n'est pas rare actuellement d'avoir des circuits à plus de 200 entrées, ce qui veut dire 2^{200} vecteurs de test, le temps de test se compte alors en milliers d'années de temps de calcul alors que les tests après fabrication, en-ligne ou de maintenance ont des exigences de rapidité.

Les modèles de fautes servent à traduire le comportement d'une faute. Ils permettent de réduire considérablement le nombre de vecteurs de test. La question se pose du « bon » modèle. Plusieurs niveaux de définition sont possibles, du plus proche du circuit physique, au plus proche des spécifications du circuit. Ces différents niveaux sont maintenant abordés.

3.3.2 Les modèles de fautes structurelles.

Un classement des modèles de fautes couramment utilisé se fait selon l'effet qu'elles ont au niveau électrique (on considère alors les transistors et les connexions entre transistors) puis au niveau logique (on prend en compte, dans ce cas, une représentation de la structure sous forme de porte logique). Les modèles de fautes les plus couramment rencontrés en technologie MOS, selon la littérature [Fritzemeier1989], sont :

- **stuck-at** : collage permanent à 0 ou 1 d'une ligne de connexion. Ce type de faute se traduit par une erreur combinatoire d'un circuit combinatoire et est donc assez facile à détecter ;
- **stuck-open** : collage d'un transistor à l'état bloquant. Ce type de fautes se traduit par un comportement séquentiel du composant combinatoire. Il peut donc ne pas être détecté par un seul vecteur de test et réclame une séquence de vecteurs, habituellement 2, le premier étant considéré comme un vecteur d'initialisation. Il est à noter que le test de détection doit se faire à fréquence nominale de fonctionnement, sans quoi la faute peut passer inaperçue.
- **stuck-on** : collage d'un transistor à l'état passant permanent. Ce type de fautes se traduit par des valeurs paramétriques des tensions, le transistor passant se comportant comme une résistance. Sur un réseau de transistors, cela entraîne une conséquence ennuyeuse : on ne peut pas toujours détecter ces fautes par l'application de vecteurs de tests (la faute n'est pas logique) ! La solution peut alors venir du « current monitoring » ou test Iddq présenté dans la section suivante.

- **bridging-fault** : c'est une faute qui vient d'un court-circuit entre deux ou plusieurs lignes de connexion d'un circuit. On peut les classer en deux catégories, selon le comportement séquentiel ou non de la faute. On se rapporte ainsi aux cas précédents.
- **delay-fault** : des impuretés dans la structure, des défauts de masque ou de légères variations extérieures au circuit peuvent entraîner des retards sur les temps de montée ou de descente des signaux sur certaines lignes d'interconnexion. Ces fautes se traduisent par des erreurs de temps de transfert dans un circuit. Pour les détecter, on aura donc besoin de vecteurs de tests envoyés dans le circuit en fonctionnement à fréquence normale.
- **parametric-fault** : c'est un modèle des fautes dues à des déviations des paramètres de certains transistors par rapport aux spécifications d'origine. Le modèle de « stuck-on » fait parti des fautes paramétriques. Une accumulation de ces fautes peut entraîner une erreur dans la structure. Les fautes paramétriques pouvant varier en fonction de la fréquence de fonctionnement. Leur test demande donc à nouveau un fonctionnement à la fréquence nominale.
- **soft-fault** : c'est une faute temporaire due à un événement extérieur unique, comme un ion lourd. Leur effet est une faute unique très localisée entraînant une erreur logique qui peut être du même type que pour les fautes paramétriques ou de retard.
- **byzanthine-fault** : c'est une faute qui n'est pas modélisée par un des modèles précédents. Cette classe forme les fautes complexes qui sont souvent très difficiles à détecter.

La classe de fautes qui fût étudiée en premier et qui reste la plus couramment utilisée est le modèle de collage. Plusieurs algorithmes (D-algorithm, PODEM, FAN [Fritzemeier1989]) ont été développés et sont actuellement utilisés dans des générateurs automatiques de test (ATPG). Toutefois, l'utilisation de ce modèle de fautes est remise en cause : 99 % de couverture des fautes de collage est-il meilleur que 90 % de couverture de fautes fonctionnelles [Maxwell1991] ? Il est maintenant clair, pour la communauté scientifique, que ce modèle est insuffisant, quoique encore largement utilisé dans le domaine industriel. Ainsi, le taux de couverture des fautes de collage constitue une référence et il est nécessaire de montrer qu'une technique de test permet, au moins, de détecter un grand nombre de ces fautes.

D'autres modèles de fautes structurelles ont été utilisés. Les difficultés sont alors de réduire la longueur des vecteurs de test afin d'obtenir un temps de test raisonnable et de trouver un moyen efficace de génération des vecteurs de test qui permette de réaliser le test à fréquence nominale. La difficulté de ce dernier point a poussé à inventer de nouvelles techniques de test dont nous parlerons dans la section suivante.

3.3.3 Les modèles de fautes fonctionnelles

Dans le modèle de fautes fonctionnelles, l'idée est que ce qui importe est que la fonction du composant soit la bonne.

Un exemple de modèle de fautes fonctionnelles des plus simples est le suivant : « toute faute qui change la fonction combinatoire du circuit en toute autre fonction combinatoire ». De ce modèle, les

comportements séquentiels sont exclus. On peut toutefois prouver qu'il inclut les fautes de collage et est donc plus efficace que les tests utilisés pour ces dernières (voir chapitre 3). Ce modèle peut se heurter à des explosions combinatoires pour assurer un bon taux de couverture. Il n'est en fait réellement efficace que sur des structures régulières comme celles des additionneurs ou multiplieurs.

Citons enfin le cas des mémoires vives de type Random Access Memory (RAM) pour lesquelles des modèles de fautes fonctionnels sont bien admis dans la communauté scientifique. Ces modèles sont repris dans le chapitre 3.

3.3.4 Les modèles de fautes de haut niveau

Les modèles de fautes structurelles peuvent être vus comme un modèle de faute très proche de la réalité physique, alors que le modèle de fautes fonctionnelles peut être vu comme un niveau d'abstraction juste supérieur, au niveau des blocs RTL.

Des niveaux d'abstraction plus importants peuvent être définis, qui permettent de réduire le nombre de vecteurs de test. Cette approche est intéressante de ce point de vue mais pose le problème de l'éloignement des défauts physiques. Cela le rend inefficace à prendre en compte la dimension physique de la réalisation matérielle. Ils ne peuvent donc être utilisés que comme une aide à l'amélioration de la testabilité d'une structure.

3.3.5 Conclusion

Les modèles structurels sont plus proches de la réalité physique, et donc souvent plus pertinents. D'un autre côté, le modèle comportemental permet de réduire le nombre de vecteurs de test et d'utiliser les mêmes vecteurs de test dans toute la phase de conception. Il est malheureusement trop éloigné de la couche physique, ce qui met en doute son efficacité. Le modèle fonctionnel, quant à lui, se situe entre ces deux précédents modèles : il est plus proche du modèle physique que le modèle comportemental tout en pouvant être utilisé assez tôt dans la phase de conception.

Il n'existe pas de panacée pour le test, et chaque composant est un problème à part entière. Il est heureusement possible de résoudre le problème du test pour certaines fonctions des circuits intégrés comme nous le verrons dans le chapitre 3.

3.4 Contrôlabilité et observabilité.

Par rapport à un modèle de fautes, il faut, en entrée du système ou du composant, appliquer un vecteur de test capable de sensibiliser la faute, c'est la contrôlabilité. De même, il est nécessaire en sortie du système ou du composant, de pouvoir observer l'effet de la faute, c'est l'observabilité.

Ces problèmes peuvent être assez complexes lorsqu'on cherche à tester un composant isolé. Ils deviennent prépondérants lorsqu'il s'agit de tester un circuit enfoui dans un système. En effet, un tel circuit peut être très difficile d'accès direct si rien n'a été prévu pour faciliter la mise en oeuvre du test. Les problèmes de contrôlabilité et d'observabilité peuvent ainsi se retrouver à tous les niveaux du système.

3.5 Conclusion

Un modèle de fautes n'est qu'un modèle et ne saurait donc être un parfait reflet de la réalité. Toutefois, il doit être suffisamment représentatif de celle-ci pour qu'on puisse avoir une certaine confiance dans un test basé sur celui-ci. Le modèle de collage qui est actuellement le plus utilisé n'est pas le modèle le plus représentatif et il convient alors de chercher un modèle de faute plus adapté. Ce problème est fondamental si on veut que les résultats de couverture de fautes aient une vraie signification. En pratique, toutefois, nous nous référerons au modèle de collage qui reste la référence dans le domaine industriel pour le calcul de la couverture de fautes.

La partie suivante donne un rapide tour d'horizon des méthodes de test actuelles. Le but de cette présentation est de permettre une comparaison des méthodes par rapport aux caractéristiques principales des calculateurs SIMD et multi-SIMD. Cette comparaison prendra également en compte le but d'amélioration de la fiabilité dont la méthodologie générale a été décrite en conclusion du chapitre 1.

4 Les méthodes de test

4.1 Introduction : limites des méthodes de test « traditionnelles »

Jusqu'à la fin des années 80, mis à part de notables exceptions (LSSD d'IBM, BILBO, scan set,...), le test n'intervenait souvent qu'à la fin du processus de conception. Il consistait alors à forcer certaines entrées du circuit à un état déterminé et à vérifier que les sorties correspondaient aux spécifications. Cette approche n'est actuellement plus valable pour plusieurs raisons :

- les composants sont de plus en plus intégrés et de plus en plus complexes. Un grand nombre de nœuds internes ne peut plus être contrôlé avec les seules entrées/sorties primaires. Cela entraîne une chute considérable du taux de couverture ;
- ces techniques ne permettent pas de contrôler le circuit à sa fréquence nominale de fonctionnement ;
- au niveau supérieur, des techniques d'intégration comme les MultiChip Modules (MCM) et les fortes densités des cartes électroniques réclament un test hiérarchisé et structuré.

Devant cette augmentation très rapide de la complexité, il est nécessaire de rendre le test plus automatisé, donc de l'inclure au niveau des outils de CAO. Un groupe de travail et de normalisation, le JTAG (Joint Test Action Group) a ainsi donné naissance à deux normes appelées Boundary Scan IEEE 1149.1 [IEEE1990], au niveau circuit intégré et MTM-Bus IEEE 1149.5 [IEEE1995] au niveau système. L'approche utilisée dans ces deux normes est d'inclure des blocs matériels facilitant le test à l'intérieur même du circuit intégré pour la première et entre les différents composants du système pour la seconde. Ces deux normes font partie du domaine plus vaste de la conception pour la testabilité ou DFT (Design For Testability). D'autres méthodes vont plus loin et choisissent d'intégrer la procédure de test dans le circuit, c'est le BIST (Built In Self Test). D'autres méthodes sont basées sur la mesure de courant, c'est l'Iddq ou permettent de contrôler le circuit pendant son fonctionnement, c'est l'auto-contrôle. Ces différents domaines ont connu un essor important au début des années 90 qui continue

actuellement, les demandes pour le test devenant plus fortes au fur et à mesure que la technologie progresse.

Toutefois, on estime en général que le test a un échelon de retard par rapport aux outils de synthèse des circuits intégrés qui ont, eux-mêmes, un retard par rapport aux besoins de conception. La course pour combler ce retard est lancée.

Dans la suite de ce chapitre, les principales techniques de test, adaptées aussi bien au test de fabrication qu'au test en ligne et qui constituent l'état de l'art du domaine sont présentées.

4.2 Conception pour le test (DFT)

Le but des méthodes de conception pour le test est d'améliorer la contrôlabilité (facilité à appliquer le test) et l'observabilité (facilité à lire les résultats du test) des nœuds internes des circuits à forte intégration en tenant compte des problèmes de test lors de la conception du circuit. Les premières méthodes utilisées ont été des méthodes ad hoc. Celles-ci se sont ensuite orientées vers la conception de structures générales de test pouvant être automatisées par CAO. Les méthodes actuelles insèrent le test en général au niveau de la synthèse et de plus en plus au niveau de la description matériel en transfert de registres de la structure [Bennets 1994, TIE 1989].

4.2.1 Les méthodes ad hoc

Ce sont des méthodes basées sur une connaissance de la structure interne du circuit et sur l'expérience des ingénieurs de test. Elles ne sont donc en général pas intégrées aux outils de CAO, difficilement généralisables et les améliorations sont souvent difficilement quantifiables. Nous pouvons citer, sans être exhaustifs, l'utilisation de points de test

pour atteindre des nœuds internes difficilement observables ou contrôlables, le partitionnement du circuit avec multiplexage des entrées/sorties primaires ou les architectures basées sur l'utilisation de bus.

4.2.2 Les méthodes de balayage (scan path)

Ce sont des méthodes employées couramment et intégrées dans la plupart des outils de CAO. Elles donnent d'assez bons résultats mais ne sont pas toujours adaptées au système car elles sont très générales et complètement automatiques. Un de leurs principaux défauts est qu'elles ne permettent pas un test à la fréquence normale de fonctionnement du circuit.

Le but du scan path est de permettre d'accéder aux nœuds internes du circuit à tester tout en limitant le nombre d'entrées/sorties supplémentaires. La méthode employée est de modifier la partie séquentielle des circuits, qui est la plus difficile à tester : on remplace les bascules du circuit par des bascules dites « scan » qui permettent de choisir entre le fonctionnement normal et un fonctionnement de test (au moyen de multiplexeurs). Celles-ci sont ensuite reliées entre elles sous forme de registre à décalage (figure 3), formant un chemin ou chaîne de test (scan path). Ainsi, le test du circuit revient au test de sa partie combinatoire.

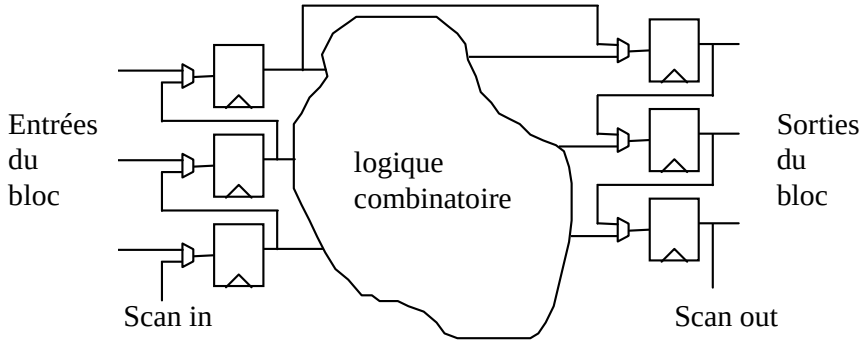


Figure 3 : principe des chemins de balayage

Il est possible de n'ajouter ces moyens de test que sur une partie du circuit, c'est le partial scan. Celui-ci réclame une analyse du circuit afin de déterminer les bascules à inclure dans le scan path. Il pose des problèmes de génération des vecteurs de test, les outils de CAO actuels n'étant souvent pas conçus pour de telles perspectives.

4.2.3 Le Boundary Scan et le MTM-Bus

Une commission, le Joint Test Action Group (JTAG) a entrepris des travaux pour la normalisation du test au niveau du circuit et du système répondant aux critères suivants :

- faciliter les accès aux broches des circuits de la carte ;
- permettre le test des interconnexions sur la carte ;
- faciliter l'utilisation des dispositifs de test intégrés aux circuits ;
- minimiser le coût en terme d'équipement de test ;
- diminuer le nombre de broches supplémentaires nécessaires à la mise en œuvre du test.

En 90, le JTAG a donné naissance à la norme boundary-scan (IEEE 1149.1) au niveau circuit intégré. Celle-ci a été suivie en 1995 de la norme MTM-Bus (IEEE 1149.5) qui porte sur la hiérarchisation du test au niveau système.

Le boundary-scan ajoute des bascules à chaque broche du circuit, de façon à pouvoir observer l'intérieur de chaque circuit. Le MTM-Bus propose un fonctionnement en maître-esclave du test des différentes hiérarchies de composants de la structure. Des précisions sur ces deux normes peuvent être trouvées dans [IEEE1990, IEEE1995].

Le boundary-scan est insuffisant pour le test interne des circuits, car il n'améliore pas l'observabilité et la contrôlabilité des nœuds internes. Il ne peut donc être utilisé comme seul test pour des circuits complexes. Enfin, le problème du fonctionnement à fréquence d'horloge normale des circuits n'est pas résolu. Le boundary-scan associé au MTM-Bus est donc une bonne solution de test pour un système, mais reste limité pour le test de l'intérieur des circuits complexes.

4.3 Le BIST

Le Built-In-Self-Test ou BIST [Agrawal1993, Mc Cluskey1985] est un moyen de tester les composants sans faire appel à une source extérieure (ATE Automatic Test Equipment). Pour réaliser cela, le générateur de test et l'analyseur de réponse sont implantés à l'intérieur de la structure (puce, carte...). Le test nécessite l'arrêt du fonctionnement normal du circuit et est considéré comme un test hors-ligne. On peut toutefois envisager l'utilisation du BIST pour un test périodique au cours du fonctionnement.

Au niveau économique, insérer du BIST augmente la surface de la puce et donc son coût. Toutefois, cette augmentation peut être contrebalancée par la facilité et la rapidité de génération des vecteurs de test. L'intérêt se trouve fortement augmenté quand il s'agit de tester un système entier où les méthodes classiques sont trop longues et inefficaces. On peut ajouter à cela que l'introduction de BIST facilite également la maintenance. De plus, et contrairement aux techniques de scan, le test peut s'effectuer à la vitesse normale de fonctionnement du circuit, ce qui en fait une technique très intéressante pour les technologies à venir.

Le BIST implique l'ajout de trois blocs supplémentaires au composant devant être testé :

- un générateur de test qui produit les vecteurs de test pour le circuit ;
- un analyseur de réponse qui vérifie que la réponse est conforme à la donnée attendue ;
- un contrôleur de test qui active les différentes phases de test.

Les solutions de génération de test couramment employées sont :

- des ROM avec microprogrammes pour un test déterministe ;
- des compteurs binaires pour des tests exhaustifs ou pseudo-exhaustifs ;
- des registres à décalage spéciaux (LFSR : Linear Feedback Shift Register) et des automates cellulaires pour du test pseudo-aléatoire ou pseudo-exhaustif.

D'autres solutions de génération de vecteurs de test adaptées à certaines parties spécifiques d'un circuit (notamment les chemins de donnée) existent. Néanmoins, l'approche la plus employée est la génération de vecteurs pseudo-aléatoires qui utilisent des LFSR, même si les automates cellulaires donnent des modèles aléatoires plus vrais. La difficulté de cette approche est l'obtention d'un taux de couverture de fautes correct pour un surplus matériel, une perte en performance et un temps de test minimaux.

Quand on applique un vecteur de test sur un circuit, on doit connaître la réponse correcte pour pouvoir effectuer la comparaison. On peut stocker cette réponse sur des ROM mais cela pose un problème de surplus matériel. Une autre solution est de compresser ou de compacter (compression avec perte de donnée) les réponses. On obtient ainsi une « signature » du circuit, sous forme d'un nombre restreint de données. La compaction entraîne des problèmes d' « aliasing », c'est à dire de perte d'information, qu'il faudra minimiser.

Les principales méthodes de compaction sont :

- le comptage de transition (nombre total de transition 0 à 1 et 1 à 0) ;
- l'analyse de signature réalisée par LFSR. La longueur de la compaction est alors la longueur du LFSR (nombre de bascules). Si la compaction se fait sur plusieurs entrées en parallèle, on parle alors de MISR (Multiple Input Shift Register) ;
- le comptage de syndrome (nombre de 1 de la séquence réponse).

Enfin, il existe certaines structures de BIST pour certains blocs spéciaux, comme les PLA, ROM et RAM [JET1994] et les structures régulières que sont les additionneurs, multiplieurs .... Ces deux points sont repris dans le chapitre 3.

4.4 Insertion de testabilité par synthèse comportementale

Généralement, les insertions de scan se font au niveau RTL. Le BIST est également introduit après la définition des différents éléments matériels. La prise en compte de la testabilité à un plus haut niveau de conception doit permettre d'améliorer la testabilité. La synthèse de haut niveau pour le test est alors requise.

La synthèse de haut niveau classique nécessite deux phases qui sont l'ordonnancement (scheduling) et l'allocation des ressources. Le but de l'insertion de testabilité à haut niveau est d'ajouter à ces deux phases des considérations de testabilité qui permettent de prédire cette dernière très tôt et d'optimiser la conception en terme de test, de performance et de surface.

Les manières d'améliorer la testabilité sont très variées. Une solution est fournie par [Papachristou1991] et concerne les chemins de donnée. La synthèse permet d'inclure des techniques de BIST dans ces chemins. Une autre solution, donnée par [Flottes and Rouzeyre1996] organise le chemin de données de telle façon qu'autant de modules que possible puissent être testés par des vecteurs de test provenant des entrées primaires. Ceci doit permettre un fonctionnement à la fréquence nominale du circuit pendant la phase de test. Ces méthodes supposent que l'ordonnancement des opérations a été effectué au préalable, ce qui limite les améliorations possibles de la testabilité.

L'ordonnancement et l'allocation sont pris en compte dans [Lee1992a, Lee1992b], mais de façon séparée et basés sur des règles simples et heuristiques d'amélioration de la testabilité. [Yang and Peng1998] prend en compte ces deux phases afin d'améliorer la contrôlabilité et l'observabilité du circuit. Ces différentes approches sont intéressantes pour la synthèse d'ASIC à partir de descriptions comportementales. Elles ne sont bien sûr d'aucune utilité lorsque l'architecture du circuit est définie par avance.

4.5 Le test Iddq

Toutes les techniques de test peuvent être mises en parallèle avec l'I_{ddq} Testing [Ferguson1990]. Cette méthode utilise la mesure du courant de puissance passif (au niveau de la masse) pour repérer les fautes du système. On remarque en effet que les fautes de transistor passant (stuck-on) et

certaines fautes entre interconnexions (bridging) créent des courts-circuits locaux dans le circuit qui entraînent une augmentation « significative » de ce courant. La méthode consiste alors à envoyer des vecteurs de test dans le circuit et à comparer le courant de masse par rapport à une référence. Elle permet la couverture de certaines fautes indécelables ou difficilement décelables par les méthodes utilisant une simple vérification logique. Pour cette raison, ce test est souvent associé aux techniques de scan ou de BIST. Le test logique permet alors de couvrir la majorité du circuit, et quelques vecteurs judicieusement choisis permettent d'appliquer l'Iddq et donc de compléter la couverture de fautes.

L'inconvénient majeur de cette technique est la difficulté du choix des vecteurs de test et la lenteur des capteurs qui empêchent l'utilisation du test à vitesse de fonctionnement normal. L'approche Built-In Current Sensor (BICS) permet alors d'intégrer le capteur directement sur la puce, permettant un test à la fréquence de fonctionnement hors-ligne, mais également en-ligne.

Les inconvénients du test Iddq résident dans la difficulté de mise en œuvre (il est nécessaire de mesurer un courant très précisément sans agir sur ce courant). Cette difficulté devient de plus en plus importante avec les progrès technologiques qui utilisent des sources en tension de plus en plus faibles et des courants également plus faibles. L'insertion de circuits de contrôle de courant à l'intérieur même de la puce peut également poser des problèmes de perte en performance due à la baisse de tension induite par le circuit de mesure.

4.6 L'auto-contrôle

L'auto-contrôle ou self-checking est utilisé dans un but de test lors du processus normal de fonctionnement appelé également test « en ligne », en utilisant les données du système comme vecteurs de test. Cette technique est adaptée à la couverture des fautes transitoires du circuit.

La méthode est la suivante : les données en entrée du circuit sont codées de façon à faire apparaître une redondance d'information qui peut alors être utilisée pour révéler les fautes du circuit. La structure est constituée du bloc fonctionnel sous test et d'un contrôleur de ce bloc. Ce dernier peut également être fautif et il est donc nécessaire de le concevoir de façon particulière.

Le but à atteindre de manière à rendre le circuit « sûr en fonctionnement » (voir Annexe 1) est celui du « totalement auto contrôlable » (TSC : Totally Self Checking), qui s'exprime ainsi : « la première sortie erronée du bloc fonctionnel doit provoquer une indication d'erreur en sortie du contrôleur ». Cela implique des propriétés spéciales des circuits logiques et des contrôleurs plus une hypothèse de fréquence d'erreur. Notre but, ici, est de présenter brièvement ce domaine complexe. Pour plus de précision, on pourra se référer, par exemple, à [Nicolaidis1984, Anderson1971, Smith1980].

4.6.1 Les hypothèse H1 et H2

L'hypothèse H1 suppose que les fautes surviennent les unes après les autres et n'est pas trop restrictive. L'hypothèse H2 porte également sur la répartition temporelle des fautes, mais a des conséquences plus importantes. Elle se définit en 2 points :

- Après l'occurrence d'une faute sur le bloc logique, il s'écoule un laps de temps suffisamment long pour que la faute soit détectée avant qu'une nouvelle faute n'apparaisse dans le bloc logique ;
- Après l'occurrence d'une faute sur le contrôleur, il s'écoule suffisamment de temps pour que la faute soit détectée avant qu'une nouvelle faute n'apparaisse dans le contrôleur ou dans le bloc logique.

Lorsqu'on utilise la méthode d'auto-contrôle, il faut donc s'assurer que l'hypothèse H2 est vérifier. Pour ce faire, une solution est d'associer ce test à un test hors-ligne périodique.

4.6.2 Caractéristiques des blocs fonctionnels et des contrôleurs en vue du test en ligne

Un circuit est « fault secure » si les sorties fausses de celui-ci sont en dehors du code de sortie. C'est à dire si ces sorties ne sont pas représentatives d'un fonctionnement normal. Pour le but TSC, un bloc fonctionnel doit être Strongly Fault Secure (SFS) : pour un ensemble de pannes donné, soit il reste « fault secure », soit ces fautes n'affectent pas le fonctionnement du circuit (panne latente).

Les qualités que doit posséder un contrôleur sont différentes. Un circuit est à code disjoint s'il transforme une entrée dans le code en sortie dans le code et une entrée hors code en sortie hors code. Pour le but TSC, un contrôleur doit être 'Strongly Code Disjoint' (SCD) : en l'absence de pannes du contrôleur, ce dernier est à code disjoint ; en présence d'une panne, ou bien le contrôleur reste à code disjoint, ou bien il existe au moins une valeur d'entrée correcte qui est transformée en un mot hors code (signalant l'erreur).

4.6.3 Les codes

Ils sont très importants car ils déterminent les modèles de fautes couverts par le test en ligne et le surplus matériel. Ils indiquent parfois le type de blocs sur lesquels ils sont applicables.

Le code le plus utilisé est le code de parité. Il consiste à ajouter un bit qui donne la parité (ou l'imparité) de la donnée. Ce code très simple est néanmoins très efficace, puisqu'en plus de détecter une faute unique, il peut détecter tout nombre impair de fautes. Très peu coûteux pour le test des interconnexions et des mémoires, il est souvent utilisé pour cet usage, notamment dans la plupart des processeurs modernes (Pentium, UltraSparc, Alpha,...).

Les codes de Berger ajoutent le complément du nombre de 0 ou de 1 du mot à la fin du mot. Il permet de prendre en compte toutes les fautes unidirectionnelles (plusieurs bits peuvent être modifiés, mais dans un seul sens 0 => 1 ou 1 => 0) de façon optimale (avec le minimum de bits supplémentaires).

Le code double-rail associe au mot son complémentaire. On peut alors couvrir tout type de fautes (fautes multiples), mais le nombre de données est multiplié par deux et la surface en silicium est plus que doublée.

Nous pouvons enfin citer des codes très différents qui sont les codes arithmétiques. Ceux-ci se conservent lors de calculs arithmétiques, comme les additions ou multiplications, ce qui les rend très intéressants pour ces structures. Les plus connus sont les codes des résidus.

4.6.4 Conclusion

L'auto-contrôle est un test en ligne qui permet de détecter immédiatement les fautes transitoires qui ne sont pas détectées par le test hors ligne. Cette détection immédiate peut être utilisée pour des mécanismes de tolérance aux fautes.

Par contre, l'hypothèse H2 limite l'utilisation de l'auto-contrôle au test en fonctionnement normal et l'interdit pour le test de fabrication. Un autre problème est le surcoût en surface de silicium qui peut être très important.

4.7 Utilisation de redondance matérielle

Cette solution est, comme la précédente, dédiée au test en-ligne du circuit. Elle utilise également de la redondance matérielle pour détecter les fautes. Cette méthode peut être vue à la fois comme une solution de test en ligne et comme un moyen de tolérer les fautes du système (voir chapitre 1). Cette dernière utilisation prime souvent sur la première.

La technique la plus couramment utilisée est celle du vote majoritaire. Les mêmes données sont entrées dans plusieurs circuits identiques ou ayant la même fonctionnalité. On contrôle ensuite que les sorties sont identiques. Cela pose le problème du contrôleur ou voteur des sorties. En effet, celui-ci est critique.

Plusieurs solutions de test en ligne sont possibles, selon le nombre de circuits en parallèle. Avec deux, on peut identifier s'il y a faute, mais pas quel circuit est fautif. La solution à trois circuits (TMR, Triple Modular Redundancy [Wakerly1976]) permet de localiser une faute si un seul circuit est fautif. C'est surtout une solution de tolérance aux fautes si le contrôleur choisit la donnée la plus représentée à la sortie des circuits. Cette technique est également appelée vote majoritaire. Elle est bien sûr très gourmande en matériel, puisque celui-ci est multiplié par plus de 3 pour un vote 2 sur 3.

La même remarque qu'au paragraphe précédent s'impose. Le vote majoritaire ne permet pas d'être sûr de la fabrication. De plus, le surcoût matériel est très important et ces solutions ne sont envisageables que pour des applications critiques. Par contre, cette approche associe de façon très efficace le test en-ligne et la tolérance aux fautes.

4.8 Conclusion

Dans cette section, 6 méthodes de test ont été présentées. Les méthodes les plus employées actuellement sont celles de balayage avec le scan path et le boundary scan. La tendance est toutefois à intégrer de plus en plus le test à l'intérieur des puces. Par ailleurs, les évolutions technologiques entraînent une plus grande sensibilité des circuits aux phénomènes extérieurs. Le problème de couverture des fautes transitoires devient donc de plus en plus important. Enfin, on remarque le retard

de l'implémentation des techniques de test au niveau de la CAO électronique, le test en étant le parent pauvre.

5 Les méthodes de test et les calculateurs parallèles intégrés

Afin de mieux appréhender l'apport des différentes méthodes de test sur les calculateurs parallèles, la figure 4 donne une représentation purement structurelle du calculateur intégré. Il est formé d'une carte électronique sur laquelle sont disposés des composants MultiChip Modules (MCM). Ces derniers contiennent plusieurs ASICs, chacun de ces ASICs comptant plusieurs processeurs.

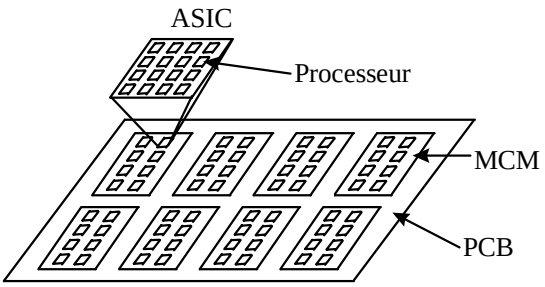


Figure 4 : structure de calculateur parallèle intégré

La méthodologie d'amélioration de la fiabilité impose deux contraintes pour le test :

- on doit avoir un test de fabrication et un test pendant le fonctionnement, périodique et/ou en ligne ;
- le test doit donner l'état de fonctionnement de chaque processeur ;

Les autres contraintes sont données par la nature même des calculateurs parallèles et, dans notre cas, par leur intégration. Pour la comparaison des différentes méthodes de test, certains critères sont donc pris par rapport à ces contraintes. Ces critères sont :

- surplus en surface très faible pour un processeur. Ces éléments sont répliqués de nombreuses fois. Par conséquent, un surplus en surface pour le test, même assez faible, peut entraîner un surplus de surface important pour le système ;
- le test d'un processeur et celui de la structure ne doit pas entraîner de baisse significative des performances. Cela signifie que chaque processeur doit être conçu avec la plus grande attention, par exemple en mettant le moins possible de multiplexeurs dans les chemins de donnée ;
- le test doit se faire à la fréquence de fonctionnement nominale. Ceci en raison des technologies agressives utilisées pour les composants le formant ;
- le test périodique doit être très court, afin de ne pas pénaliser la structure. Cela signifie qu'il doit être fait en parallèle sur chacun des processeurs.

Le tableau 1 reprend les 6 méthodes de test présentées dans la section précédente et utilise les critères qui viennent d'être définis afin de les classer.

	phase de test	fréquence de fonctionnement	Surplus matériel	Baisse en performance	Difficultés d'implémentation
DFT	fabrication	horloge de test	assez faible	assez faible	requiert un ATPG
BIST	fabrication ou périodique	nominale	faible à moyen	faible à moyen	choix de la méthode
insertion de testabilité à haut niveau	fabrication ou périodique	horloge de test ou nominale	incalculable	incalculable	passe par des outils CAO
Iddq	fabrication, périodique ou en-ligne	horloge de test ou nominale	assez faible	moyenne	définition des vecteurs de test et du courant de référence
Auto-contrôle	en-ligne	nominale	moyen à important	faible à important	modification structurelle des blocs
redondance matérielle	en-ligne	nominale	très important	faible	conception du contrôleur

Tableau 1 : caractéristiques des méthodes de test par rapport aux contraintes de fiabilité et d'intégration des calculateurs parallèles

Le critère de fonctionnement à fréquence nominale écarte les méthodes de DFT comme le scan, qui, par ailleurs, induisent des multiplexeurs dans certains chemins qui peuvent s'avérer critiques dans notre cas. Les méthodes d'insertion de testabilité au niveau comportemental sont également exclues car on ne dispose pas d'une description comportementale mais structurelle des processeurs.

Le test Iddq est également abandonné, ceci pour deux raisons principales : tout d'abord la difficulté de mise en œuvre dans notre cas. En effet, chaque processeur doit être testé à part, puisqu'on désire connaître leur état. Il est alors difficile de définir avec précision le courant de masse processeur par processeur sur la couche physique d'un ASIC. De plus, cela nécessiterait une conception très particulière du circuit qui n'est pas voulue dans notre cas. La deuxième raison vient de la perte en performance de la structure due à l'intégration d'un capteur de courant sur la puce, ce dernier entraînant une baisse de tension.

Enfin, La redondance matérielle entraîne des surplus trop importants. Même si dans notre cas, la redondance matérielle est inhérente à la structure, il paraît difficile de l'utiliser pour faire du vote majoritaire sans entamer fortement les performances ou augmenter fortement le coût matériel.

Les techniques de BIST et d'auto-contrôle sont celles qui correspondent le mieux aux critères définis. Le BIST est très intéressant puisqu'il peut permettre de faire les phases de test de fabrication et périodique. Ce dernier point est également important pour le respect de la règle H2 pour l'auto-contrôle. Les dangers de ces deux méthodes résident dans le surplus matériel ainsi que dans les pertes en performance qu'elles peuvent engendrer. Ce sont ces 2 méthodes qui sont reprises et adaptées aux calculateurs parallèles dans le chapitre 3.

6 Conclusion

Les différents modèles de fautes, les différentes phases de test ainsi que les principales méthodes de test ont été présentées. A la suite de la comparaison des différentes méthodes de test, le BIST et l'auto-contrôle ont été retenus comme les méthodes les mieux adaptées à nos critères.

Le chapitre suivant traite du test des calculateurs parallèles de type SIMD ou multi-SIMD, en se focalisant plus particulièrement sur le test d'un processeur. Ce dernier doit en effet être optimisé en raison du grand nombre de processeurs qui composent la structure.

Chapitre 3 : test des calculateurs parallèles SIMD ou Multi-SIMD

1 Introduction

Le but fixé par la méthodologie générale d'amélioration de la fiabilité (voir chapitre 1) concernant le test est de mettre au point une procédure adaptée aux calculateurs parallèles de type SIMD ou multi-SIMD permettant de tester le circuit en fin de fonderie, mais également lors de fonctionnement. Ce dernier test doit pouvoir s'effectuer de façon permanente, « en-ligne » ou de façon périodique, selon les demandes en fiabilité du système ou de l'application.

Pour certaines des applications visées, tel le traitement d'images en temps réel (à la fréquence vidéo), il n'est pas nécessaire de couvrir les fautes transitoires. En effet, une faute n'affectant que temporairement un circuit ne dégrade que quelques images (et en général sur une surface réduite), sans avoir un effet visuel perceptible. Un test en-ligne, qui peut s'avérer coûteux en performance et surface de silicium, est alors inutile. Par, contre, ce même test apparaît nécessaire lorsque le calculateur est dédié à des applications demandant de longues heures de calcul, où chaque résultat influe ou peut influer sur les calculs suivants.

La principale difficulté de toute méthode de test consiste dans le choix du modèle de fautes sur lequel s'appuyer (voir chapitre 2). Certaines parties, comme les mémoires, ont des modèles de fautes bien définis et en général bien admis dans la communauté scientifique. Par contre, les modèles de faute pour la logique générale sont connus mais souvent difficiles à couvrir, ou requérant un grand nombre de vecteurs de test. Ce dernier point pose donc des problèmes lorsqu'on désire utiliser ce test pour faire du test périodique. Par ailleurs, des méthodes pour couvrir de « nouvelles » fautes comme les fautes de délais émergent. La question suivante se pose alors : « quel est le modèle de fautes réalistes qui permet le meilleur compromis avec la durée du test ? ». Un élément de réponse à cette question est que le modèle doit englober, dans la mesure du possible, des fautes représentatives des dernières technologies. Ce dernier point implique que :

- Il est nécessaire d'effectuer le test à la fréquence nominale du circuit ;
- le test, quel qu'il soit, doit également couvrir un maximum de défauts non modélisés par les modèles de faute.

D'après le premier point, il est légitime de penser que le test intégré ou BIST, qui permet un fonctionnement à la fréquence nominale du circuit, est une voie d'avenir.

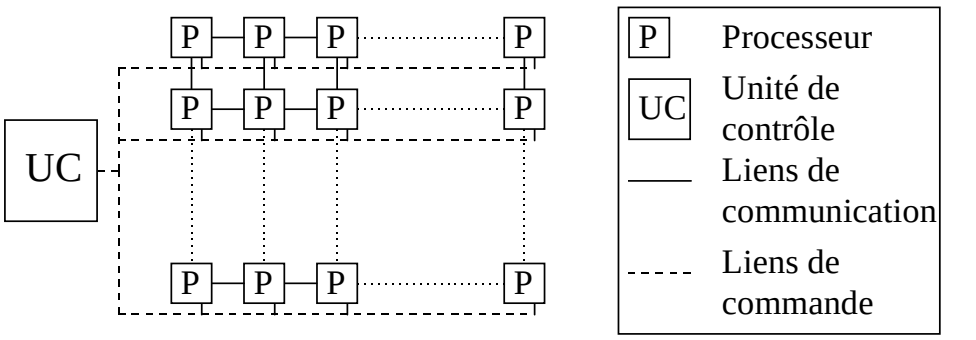
Le deuxième point, traité par exemple dans [Maxwell1991], nous indique qu'il est souvent inutile de chercher à traiter des fautes très difficiles à détecter sur certains modèles, car l'effort fourni n'a pas le bénéfice escompté. Il semble en effet que les défauts non modélisés soient généralement nombreux. Dans ces conditions, améliorer le test des fautes difficiles revient souvent à augmenter très faiblement la couverture effective des défauts.

D'autre part, les méthodes de test adaptées pour chacune des parties d'un circuit sont très différentes les unes des autres. Par conséquent, il est intéressant de partitionner le circuit, et de prévoir un test au niveau de chaque bloc fonctionnel.

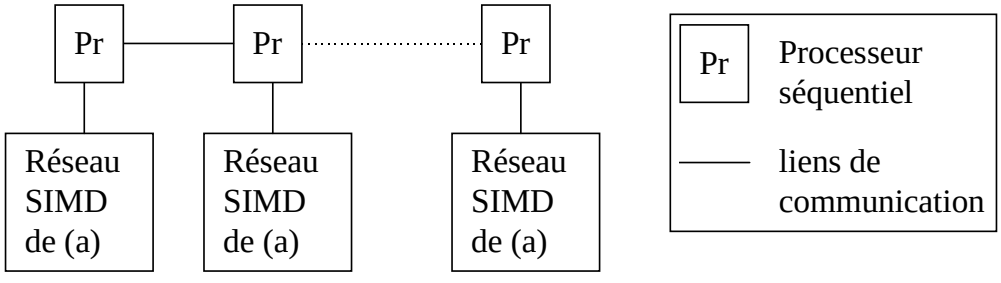
A partir de ces éléments, la section suivante traite de la méthodologie de test que nous allons suivre pour la définition d'une stratégie de test cohérente avec l'ensemble des problèmes rencontrés pour les calculateurs parallèles de type SIMD ou multi-SIMD.

2 Méthodologie de test

Un calculateur parallèle SIMD ou multi-SIMD est composé de trois parties (figure 1) : les processeurs, éléments assez « simples » répétés un grand nombre de fois ; les interconnexions entre ces éléments et la partie contrôle du réseau et des processeurs. Cette dernière peut être constituée soit d'une simple machine d'état, soit d'un processeur séquentiel évolué qui peut alors être un processeur CISC ou RISC, un DSP, ...



(a)



(b)

Figure 1 : (a) les calculateurs SIMD sont composés de trois parties : l'unité de contrôle, les processeurs et le réseau de liens de communication reliant ces processeurs (b) un exemple de réalisation de calculateur Multi-SIMD. Dans ce cas, seuls les processeurs séquentiels peuvent communiquer entre eux

Le test de l'élément de contrôle, partie délicate et critique du système, n'est pas traité. Nous supposons en effet que cette partie est constituée, dans notre cas, d'un processeur séquentiel

existant, de type RISC ou CISC, ou d'un DSP. Celui-ci doit permettre de réaliser rapidement les parties séquentielles des applications en profitant des dernières évolutions technologiques. Des techniques de test particulières sont associées à chacun de ces processeurs. En général, ils possèdent déjà des moyens de test sous forme de boundary scan (norme IEEE 1149.1), et l'absence d'information sur leur structure interne limite fortement les possibilités d'en effectuer un test différent de celui proposé par le constructeur. D'autre part, cet élément étant prépondérant dans la structure tout en n'en représentant qu'une faible proportion, il est possible d'imaginer l'utilisation de méthodes de vote majoritaire (comme le TMR : Triple Modular Redundancy) afin d'atteindre un haut niveau de fiabilité lors du fonctionnement du système. Il est à noter que d'autres solutions, moins gourmandes en matériel, mais peu convaincantes [Avizienis1997], peuvent être envisagées. C'est, par exemple, l'utilisation d'un module contrôlant en permanence la machine de contrôle d'exception (Machine Check Architecture) ainsi que les codes correcteurs d'erreur (limités aux parties mémoire) pour le Pentium III. Cela peut également être un bloc de contrôle des codes détecteurs d'erreur (également limités aux parties mémoire) pour l'UltraSparc II ou le MIPS 10000.

Par contre, le test des processeurs du tableau SIMD sera le point central de la suite : ces éléments, très nombreux, réclament un test rigoureux pour un surplus matériel et une diminution de performance minimaux en un temps de test très court pour être compatible avec le test périodique (voir chapitres 1 et 2). L'effort sur ces éléments peut être important, puisque le gain apporté est proportionnel au nombre de processeurs du calculateur SIMD. Afin de rendre le test très rapide, il apparaît nécessaire de tester tous les processeurs en parallèle, ceci rendant l'utilisation du test BIST très appropriée.

La suite de ce chapitre est organisée ainsi : nous nous intéressons tout d'abord au test de fabrication qui doit être compatible aux contraintes du test périodique. Cette première partie est suivie de la description d'une solution pour réaliser un test concurrent (dit « en-ligne ») du circuit, test nécessaire si on veut se prémunir contre les fautes transitoires. Nous donnons enfin la méthodologie de test des interconnexions entre processeurs.

3 Test « hors-ligne » d'un processeur

3.1 Introduction

Les processeurs du tableau SIMD peuvent être réalisés de plusieurs façons :

- par une description via un langage de spécification matériel comme VHDL ou Verilog, suivie d'une synthèse logique ;
- à partir de circuits prédiffusés comme les FPGA. Les processeurs perdent dans ce cas beaucoup de leur vitesse d'exécution mais gagnent en souplesse de conception et d'utilisation ;
- à partir de circuits sous forme de propriétés intellectuelles (IP : Intellectual Properties) qui sont alors des cœurs de processeurs existants. Ces IP peuvent être de trois sortes : description comportementale du composant (IP soft) ; description RTL synthétisable du composant avec contraintes de placement-routage (IP synthétisable) ; bloc matériel placé, routé et optimisé selon une filière technologique (IP hard).

L'utilisation de FPGA ne permet pas une intégration suffisante des processeurs, ils ne sont donc pas retenus. Nous considèrerons, pour la conception du processeur, la réalisation par modélisation VHDL ainsi qu'à partir des deux premières sortes d'IP (soft et synthétisable). La troisième catégorie d'IP n'est pas retenue, car la souplesse pour la conception pour le test est nulle.

Afin de prendre en compte ces 3 types de réalisation, le modèle retenu est une structure incluant des schémas généraux, et pouvant donc être adaptée plus facilement à des cas particuliers. Le processeur sur lequel nous nous appuyons pour la suite est décrit figure 2. Nous supposons connus les différents blocs et les interconnexions entre les blocs (schéma structurel à haut niveau). Par contre, aucune hypothèse n'est faite sur l'implémentation matérielle de chaque bloc. Aussi, la méthode de test ne devra demander qu'une faible modification de l'écriture de la description matérielle du processeur.

Le processeur présenté a été simplifié de façon à ne prendre en compte que les parties essentielles d'un processeur de tableau SIMD. Toutefois, comme nous le verrons par la suite, le test peut être étendu à des structures plus complexes.

Le processeur présenté figure 2 inclut un banc de registres, une mémoire RAM, une unité arithmétique et logique (ALU), un multiplieur (*****), ainsi qu'un contrôleur du réseau et un bloc de décodage/séquencement des micro-instructions provenant de l'unité centrale (UC).

Deux éléments importants du processeur sont plus précisément étudiés. Il s'agit des mémoires (registres et RAM) et du chemin de données (UAL, multiplieur et différents multiplexeurs). Des indications sont ensuite données pour les autres parties du circuit. Enfin, nous faisons la synthèse des différents tests sous la forme de résultats de synthèse de circuits et de simulation de fautes de collage.

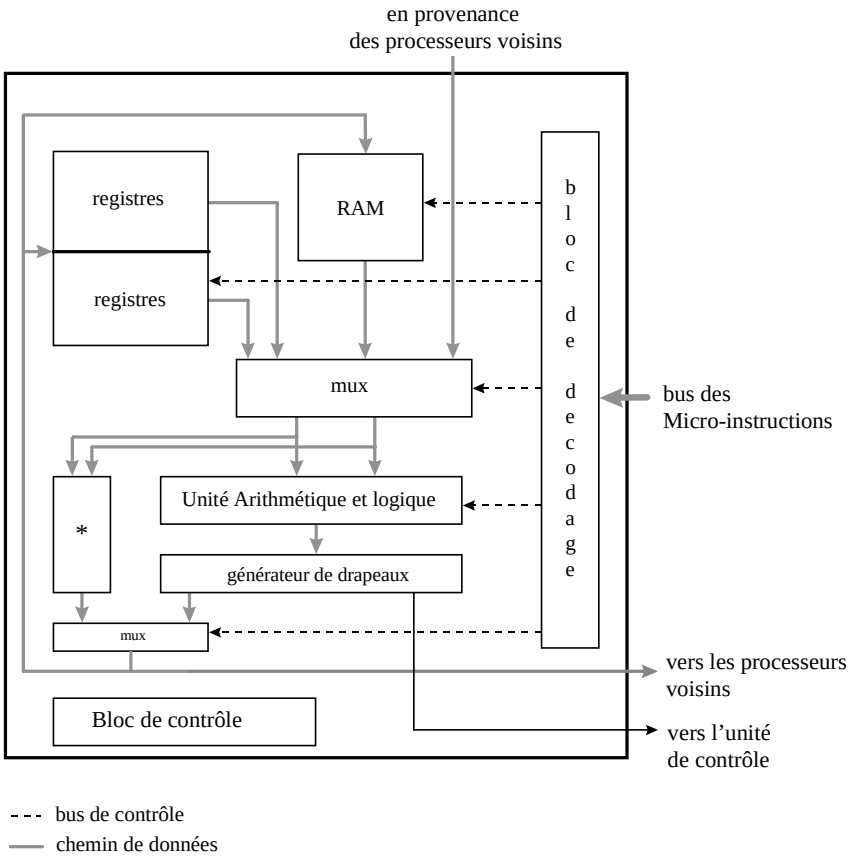


Figure 2 : processeur simplifié d'un tableau SIMD. Il est constitué d'un banc de registres associé à une mémoire RAM qui alimentent une unité arithmétique et logique ainsi qu'un multiplieur. Une unité de décodage permet d'assurer le déroulement des micro-instructions

3.2 Test des mémoires

3.2.1 Introduction

Les éléments de mémorisation (RAM, ROM, registres...) sont les éléments les plus « faciles » à tester et ont été largement étudiés au niveau du test en raison de leur forte densité d'intégration qui les rend plus susceptibles de comportements fautifs. Les modèles de fautes pouvant affecter ces circuits sont bien connus, et de nombreux algorithmes ont été développés afin de couvrir chacun d'entre eux. Toutefois, ces algorithmes sont, la plupart du temps, adaptés au test des mémoires orientées bits (dont chaque bit peut être lu et écrit séparément). Or, les mémoires de nouvelles générations sont orientées mots afin d'augmenter leur bande passante. On parle alors de Word Oriented Memories (WOM). Le test de ces structures est généralement réalisé à partir des procédures développées pour les mémoires orientées bits. Pour ce faire, on utilise plusieurs initialisations des mots de la mémoire avant l'application de la procédure de test. Cette méthode n'est pas efficace pour les fautes intra-mots comme le montrent A.J. Van de Goor et al. [Van de Goor and Tlili1998]. Ceux-ci ont proposé un algorithme de test dédié aux WOM de très bonne qualité pour palier cette difficulté.

La volonté d'inclure une phase de test périodique pour les processeurs implique en général la sauvegarde du contexte avant chaque test et le rechargement du contexte après la procédure de test. Dans le cas des calculateurs SIMD, une telle sauvegarde prendrait un temps très important et pénaliserait trop la structure.

Le concept de « test transparent » formalisé dans [Nicolaidis1992, Nicolaidis1996] permet de résoudre cette difficulté. La formulation de ce concept est la suivante : « concevoir une procédure de test permettant, en fin de test et si aucune faute n'est détectée, de retrouver en mémoire les données présentes avant l'application de la procédure de test ». Le test devient ainsi « transparent » pour le programme, qui peut redémarrer sans rechargement de contexte. Jusqu'à présent, la formulation de ce concept n'a été faite que pour des mémoires orientées bits.

Afin de résoudre le problème du test des mémoires des structures SIMD selon les contraintes qui ont été données, les concepts de test orienté mots et de test transparent doivent donc être unifiés. Pour ce faire, nous présentons les différentes mémoires, les modèles de fautes et les algorithmes courants, puis nous démontrons que le concept de test orienté WOM peut être étendu à celui de test transparent.

3.2.2 Les différentes mémoires

Les mémoires utilisées dans le processeur présenté ci-dessus sont de deux types : des mémoires à accès aléatoires de type RAM, et des mémoires à base de bascules que sont les bancs de registres. Cette dernière catégorie, beaucoup moins intégrée que la première (qui fait appel à des compilateurs de silicium spécialisés), n'est affectée que par un sous-ensemble des fautes pouvant survenir dans les mémoires RAM. Pour cette raison, nous nous focalisons uniquement sur celles-ci.

Une mémoire RAM comprend quatre éléments :

- Un tableau de cellules mémoires ;
- Un décodeur d'adresses ;
- un registre d'adresses et de données ;
- une logique de lecture/écriture.

Sa réalisation physique dépend de la technologie et de l'architecture utilisée. On distingue, en général, les RAM statiques (SRAM) et les RAM dynamiques (DRAM). Leur architecture générale est à peu de choses près la même. La mémoire est divisée en petits blocs très denses, chacun entouré d'une logique de décodage, d'amplificateurs et de circuits de pré-chargement. La différence essentielle entre les mémoires statiques et dynamiques vient de la cellule de base qui est de conception très différente. Celle des RAM statiques est constituée de 6 transistors qui forment une cellule possédant deux états stables. Elle conserve ainsi la donnée tant que les transistors sont alimentés. La cellule de base des RAM dynamiques est constituée de 4 transistors formant une cellule à effet capacitif. Le contenu de celle-ci doit donc être rafraîchi périodiquement. Les mémoires DRAM comportent donc un nombre de transistors plus faible que les SRAM. Elles sont également moins sensibles aux conditions extérieures du fait de leur rafraîchissement périodique.

Des modèles de fautes structurels ont été établis pour les deux sortes de RAM. Selon le type de mémoire, certaines fautes seront alors plus susceptibles d'apparaître que d'autres.

Les RAM orientées mots peuvent être organisées de deux façons différentes [JET1994] : mots adjacents les uns aux autres ou mots entrecroisés. Dans ce dernier cas, les M bits du mot sont physiquement séparés par des bits d'autres mots. Ces mémoires ne nécessitent qu'un sous-ensemble des tests nécessaires par rapport aux mémoires formées de mots adjacents. Par conséquent, seul le cas des RAM à mots adjacents est considéré.

3.2.3 Modèles de fautes

En principe, chacun des éléments d'une mémoire doit être testé de façon différente. En pratique, on ne s'intéresse qu'aux fautes des cellules. On peut en effet déterminer que les fautes sur les autres éléments se répercutent sur les cellules et se détectent donc facilement lors du test de ces dernières.

Deux hypothèses sont utilisées pour le développement des modèles de fautes et des algorithmes de tests :

- hypothèse de faute unique : un test détectant ce type de fautes détecte également un grand nombre de fautes multiples.
- hypothèse de lecture sans faute due à des considérations pratiques : ces fautes ne se produisent pratiquement jamais pour des raisons de technologie.

Les modèles de fautes sont maintenant décrits au travers de deux grandes classes. La première n'implique qu'une seule cellule alors que la seconde met en cause plusieurs cellules voisines.

3.2.3.1 Les fautes sur une cellule mémoire unique

On distingue :

- Les fautes de collage (Stuck-At Fault ou SAF). Une cellule est bloquée à l'état 0 ou 1 ;
- Les fautes de transition (Transition Faults ou TF). Ce sont les fautes duales des SAF. On ne considère plus l'état, mais la transition d'état de la cellule. La transition 0 vers 1 ou 1 vers 0 ne s'effectue pas ;
- Les fautes de rétention de donnée (Data Retention Faults ou DRF). Ce sont des fautes dues aux caractéristiques électriques des transistors composant la cellule. La donnée écrite dans la cellule peut être perdue après un certain temps.

Ces fautes, qui impliquent une seule cellule, peuvent être traitées de la même façon, que l'on soit en présence d'une mémoire orientée bits ou mots. Des algorithmes classiques, dont quelques exemples sont donnés dans la suite, peuvent alors être utilisés.

3.2.3.2 Les fautes d'influence entre cellules

Les fautes d'influence entre deux cellules peuvent être réunies sous la notion de fautes de couplage (CF : coupling faults).

Définition : une paire de cellules est dite couplée si une transition ou l'état de l'une d'elle entraîne le changement du contenu de la seconde.

On distingue :

- Les fautes de couplage *idempotentes* (CFid). La transition d'une des cellules (dite « agresseur ») force le contenu de l'autre à 0 ou à 1 ;
- Les fautes de couplage d'*inversion* (CFin). La transition d'une des cellules force l'autre cellule à s'inverser ;
- Les fautes de couplage par *influence écriture/lecture* (ou disturb CF, CFdst). La lecture ou l'écriture d'une des cellules entraîne un changement d'état de l'autre cellule ;
- Les fautes de couplage d'*état*. Une cellule est forcée à une certaine valeur x si l'autre cellule est dans un état y.

A ces modèles de fautes on peut ajouter la notion de faute liée :

Définition : Une faute est dite *liée* si elle peut influencer le comportement d'autres fautes.

Des fautes liées peuvent se masquer. Il est donc nécessaire de prévoir des tests spéciaux pour leur couverture.

Il faut noter que toutes les fautes de couplage n'apparaissent pas forcément dans les mémoires, selon la technologie employée pour leur réalisation. On dit alors que l'écriture domine la faute de couplage.

Lorsque plus de 2 cellules sont en cause dans l'apparition d'une faute, on parle de fautes d'influence (c'est une extension des fautes de couplage) :

Définition : une cellule mémoire subit des fautes d'influence si son état est influencé par une succession de 0 et 1, transitions de 0 à 1 ou de 1 à 0, ou par l'état d'autres cellules.

L'ensemble des cellules qui influencent la première est appelé son voisinage (c'est un voisinage physique). Le test sûr et total de ces fautes est illusoire, car il faudrait une longueur de test en $O(2^N)$ (N étant le nombre de cellules mémoires) pour les couvrir ! On est donc obligé de considérer un voisinage réduit pour chaque cellule. Les fautes d'influence sont dites statiques lorsque la cellule de base est influencée par un modèle de 0 et 1 dans son voisinage et dynamique lorsque la cellule de base est influencée par un changement des états de son voisinage.

Le modèle de fautes « line and column-weight sensitive » est un cas particulier du modèle précédent. Il est basé sur des considérations technologiques : on a en effet constaté que le contenu de la cellule peut être plus facilement influencé par les cellules dans son rang ou sa colonne physique car ces cellules sont connectées électriquement. L'avantage de ce modèle est que le test permettant de

détecter ce type de fautes détecte également la plupart des autres fautes d'influence. L'inconvénient majeur des algorithmes déduits de ce modèle est leur longueur qui est au minimum en $O(N^{3/2})$.

Afin de limiter la durée du test des mémoires (de façon à être compatible avec un test périodique), et pour des raisons d'ordre technologique (la détection des fautes de couplage permettant de couvrir un grand nombre d'autres fautes plus complexes), on ne considérera dans la suite que les fautes de couplage entre deux cellules.

3.2.4 Algorithmes courants

Les algorithmes présentés ici comprennent tous une série de séquences d'écritures suivies de lectures des cellules mémoires. Ils sont tous d'ordre N, N représentant le nombre de cellules mémoires, ceci afin de satisfaire à la contrainte de temps du test périodique. Ils sont classés du plus simple au plus complexe et sont donnés pour des mémoires orientées bits. Pour plus de détails sur chacune des méthodes, et notamment pour leurs preuves, on peut se référer à [JET1994].

Le test de *scrutation mémoire* (memory scan test)

C'est le test le plus trivial. Il consiste à écrire 0 dans chaque cellule, à lire la cellule, à y écrire 1 et à la lire à nouveau. C'est un test très court de longueur 4N mais il donne un très mauvais taux de couverture puisqu'il ne détecte même pas les fautes de collage du décodeur.

Le test *MATS +*

C'est un des tests les plus utilisés car l'algorithme est simple même si le taux de couverture est assez moyen. On initialise toutes les cellules à 0 puis on « scrute » les mémoires en ordre ascendant puis descendant. La cellule est lue, le complémentaire de la valeur y est écrit, puis la cellule est à nouveau lue avant d'effectuer la même opération sur la cellule suivante. Le fait de scruter les mémoires les unes après les autres dans l'ordre d'adressage est appelé « marche ». Cet algorithme couvre entièrement les fautes de collage et une grande partie des fautes de couplage d'état. Il sert de base pour un grand nombre d'algorithmes plus complexes offrant une meilleure couverture de fautes. En règle générale, les algorithmes dits « de marche » sont les algorithmes les mieux adaptés pour les mémoires en raison de leur facilité d'implantation et leur très bonne couverture de fautes.

Le test de *March C-*

Ce test est basé sur 6 éléments de marche. Son taux de couverture est amélioré par rapport au MATS +, puisqu'il permet de détecter les fautes de transition, et un certain nombre de fautes de couplage, dont les fautes idempotentes, d'inversion et d'état.

Le test de *March B*

Il permet, en plus des fautes couvertes par le test de March C-, la détection de certaines fautes liées : les fautes de transition liées aux fautes de couplages idempotentes et d'inversion et les fautes de couplage d'inversion liées aux fautes de couplage idempotentes.

Le test de *March LR*

C'est un test spécialement étudié pour les fautes liées. Il détecte toutes les fautes couvertes par le test de March B ainsi que les fautes liées non couvertes par le test de March B.

3.2.5 Introduction au test transparent

Il a été montré dans [Nicolaidis1996] que les algorithmes de test pour BOM pouvaient être transformés en algorithmes transparents sans perte de couverture de fautes. Prenons le début d'un algorithme de March qui s'écrit, en test classique de la façon suivante :

$$\left[W(0)R(0)W(1)R(1)W(0)\right]\uparrow \qquad (1)$$

Où W(0) signifie l'écriture d'un 0 dans la cellule, W(1) l'écriture d'un 1 ; R(0) et R(1), la lecture qui suit l'écriture et dont le résultat correct est 0 et 1 ; $\uparrow$ signifie une scrutation mémoire dans le sens croissant.

La première écriture à 0 ne sert qu'à initialiser les cellules mémoires et peut donc être omise. Considérons que la cellule i contient la donnée a_i, on peut alors transformer (1) de la façon suivante :

$$\left[R(a_i)W(\overline{a_i})R(\overline{a_i})W(a_i)\right]\uparrow \qquad (2)$$

Dans ce premier cas, la donnée a_i est conservée dans une mémoire provisoire après la première lecture. Cette donnée est alors utilisée pour les écritures suivantes. On peut également transformer (2) de la façon suivante :

$$\left[R(a_i)W(\overline{a_i})R(b_i)W(\overline{b_i})\right]\uparrow \qquad (3)$$

Dans cette nouvelle forme de l'algorithme, la donnée a_i n'est pas conservée pour l'écriture suivante : on n'utilise plus de mémoire intermédiaire, c'est la cellule mémoire qui fournit les données pour chaque écriture. Le surplus matériel pour implémenter cet algorithme est alors moins important que celui pour l'algorithme (2).

Les algorithmes (1), (2) et (3) sont équivalents si aucune faute n'apparaît. Par contre, en cas d'occurrence d'une faute, ils ne le sont plus. En effet, une faute de collage à 0 de la cellule i entraîne pour les algorithmes (2) et (3) les valeurs suivantes :

$$\left[R(0)W(1)R(0)W(0)\right] \text{ et } \left[R(0)W(1)R(0)W(1)\right]$$

Toutefois, la première erreur R(0) est détectée, ce qui veut dire que les algorithmes (2) et (3) ont un taux de couverture identique pour cette faute. Ceci est généralisable si l'algorithme de test initial (1) couvre un modèle de fautes symétriques. Un modèle de fautes est symétrique si la transformation par changement d'état des cellules ou/et des transitions des cellules donne des fautes appartenant au même modèle. Par exemple, le modèle de fautes de couplage idempotentes sur deux cellules (i,j) et la transition (j=>not(j)) ou (i=>not(j)) est un modèle symétrique : la transformation d'une faute par changement des cellules i et j ou par changement de la transition donne une faute qui appartient

également à ce modèle. Il est facile de prouver que tous les modèles de fautes décrits au paragraphe 3.2.3 sont symétriques.

Il est possible, pour les algorithmes de test couvrant des modèles de fautes non symétriques de rendre l'algorithme symétrique de façon à conserver le même taux de couverture. Prenons l'exemple très simple du morceau d'algorithme $\left[W(1)R(1)W(0)\right]\uparrow$. Sans aucune supposition sur les fautes couvertes par ce module de March, nous pouvons le rendre symétrique par l'application de deux modules de March $\left[W(\overline{a_i})R(\overline{a_i})W(a_i)\right]\uparrow$ et $\left[W(a_i)R(a_i)W(\overline{a_i})\right]\uparrow$. Nous avons alors :

a_i	$\left[W(\overline{a_i})R(\overline{a_i})W(a_i)\right]\uparrow$	$\left[W(a_i)R(a_i)W(\overline{a_i})\right]\uparrow$
0	$\left[W(1)R(1)W(0)\right]\uparrow$	$\left[W(0)R(0)W(1)\right]\uparrow$
1	$\left[W(0)R(0)W(1)\right]\uparrow$	$\left[W(1)R(1)W(0)\right]\uparrow$

Cela nous permet de conserver le module de March initial. La conservation des dépendances entre les modules de March (lorsque cela est nécessaire) est, par contre, plus difficile à réaliser et demande une nette augmentation de la durée de test, ce qui n'est pas compatible avec le test transparent qui a été choisi pour faire du test périodique des mémoires. Le choix d'un algorithme basé sur la couverture de modèles de fautes complets a donc été fait.

Le test de March B, qui est de longueur 17N a été choisi pour le test des mémoires. Son principal avantage est sa couverture de fautes qui est bonne par rapport à la longueur de l'algorithme. En effet, les fautes couvertes sont basées sur un ensemble de fautes symétriques comprenant toutes les fautes de couplage [JET1994]. La version de l'algorithme choisie est une version (3) où les données écrites dans la mémoire sont directement issues de la mémoire à chaque étape. Le test BOM transparent déduit de la transformation du test de March B en test transparent est ainsi le suivant :

$$\left[R(a_i)W(\overline{a_i})R(b_i)W(\overline{b_i})R(c_i)W(\overline{c_i})\right]\uparrow,\ \left[R(d_i)W(\overline{d_i})W(\overline{e_i})\right]\uparrow,$$

$$\left[R(f_i)W(\overline{f_i})W(\overline{g_i})W(\overline{h_i})\right]\downarrow,\ \left[R(j_i)W(\overline{j_i})W(\overline{k_i})\right]\downarrow \qquad (4)$$

Après avoir présenté les techniques de test transparent, nous allons maintenant étendre le concept de test transparent au cas des mémoires orientées mots. A partir de cette généralisation, le schéma BIST du test périodique et de fabrication des mémoires sera déduit.

3.2.6 Le test transparent orienté mots

Les tests des mémoires qui servent à couvrir les fautes de cellule unique pour les BOM peuvent être repris sans aucune modification pour les WOM. De même, les fautes de couplage inter-mots sont parfaitement couvertes par un test orienté bit comme le test de March B. Par contre, si l'écriture ne domine pas les fautes de couplage, le cas des fautes intra-mots n'est pas couvert. Les fondements d'un test transparent pour les fautes intra-mots des mémoires WOM, présentés en [Clermidy1998], vont maintenant être donnés.

Prenons l'exemple de 2 bits a_0 et a_1 situés sur le même mot. Le diagramme de la figure 3 montre les fautes de couplage idempotentes selon les transitions d'états des deux cellules. On remarque que les 4 transitions 00⇔11 et 01⇔10 permettent la sensibilisation de toutes les fautes de couplage idempotentes entre ces deux bits.

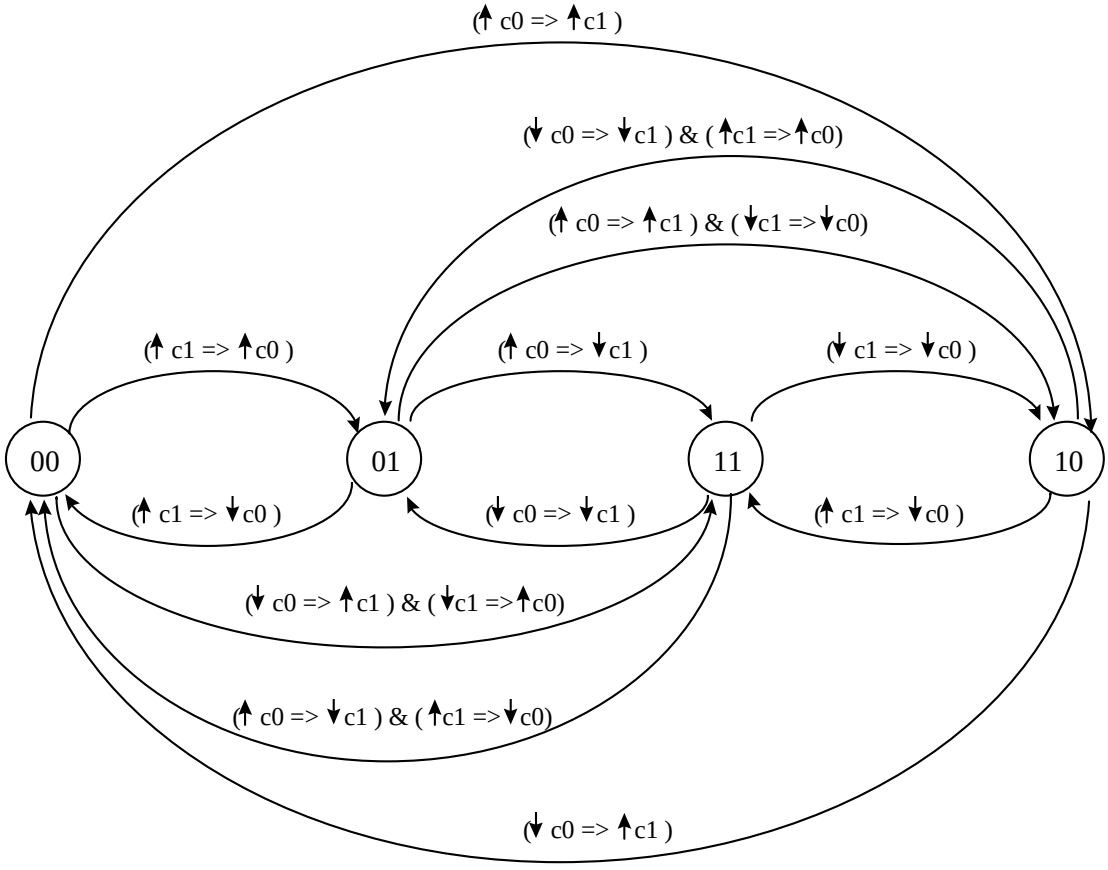


Figure 3 : diagramme d'état indiquant selon ses transitions les fautes de couplages idempotentes entre deux cellules c0 et c1 qui peuvent être activées. Les flèches indiquent les transitions. Une transition d'une des deux cellules entraîne alors une faute sur l'autre cellule

Prenons maintenant la séquence particulière :

$$S_0 = a_0a_1, \overline{a_0a_1}, a_0a_1, a_0\overline{a_1}, \overline{a_0}a_1, a_0\overline{a_1} \qquad (6)$$

Nous pouvons calculer toutes les valeurs des deux bits :

a_0a_1	S_0
00	00, 11, 00, 01, 10, 01
01	01, 10, 01, 00, 11, 00
10	10, 01, 10, 11, 00, 11
11	11, 00, 11, 10, 01, 10

Tableau 1 : séquence S_0 en fonction des données a_0 et a_1

La lecture du tableau 1 montre que quelles que soient les valeurs initiales des deux cellules, toutes les fautes de couplage idempotentes ont été sensibilisées par l'application de la séquence S_0.

Donc, si $D_0 = a_0a_1a_2a_3a_4a_5a_6a_7$ est la donnée contenue dans le mot mémoire i et si on pose $D_1 = a_0\overline{a_1}a_2\overline{a_3}a_4\overline{a_5}a_6\overline{a_7}$, alors l'application de la séquence :

$$D_0, \overline{D_0}, D_0, D_1, \overline{D_1}, D_1 \qquad (7)$$

revient à l'application de la séquence (6) pour tous les couples de bits adjacents du mot, et donc à la couverture des fautes de couplage idempotentes pour tous ces couples de bit.

De même, si on pose $D_2 = a_0a_1\overline{a_2a_3}a_4a_5\overline{a_6a_7}$ ..., l'application de la séquence :

$$D_0, \overline{D_0}, D_0, D_2, \overline{D_2}, D_2 \qquad (8)$$

revient à l'application de la séquence (6) pour tous les couples de bits à une distance 2 du mot, et donc à la couverture des fautes de couplage idempotentes pour tous ces couples de bit.

De façon identique, on peut couvrir les fautes de couplage idempotentes pour des couples de bits à une distance 4 en remplaçant dans (8) D_2 par $D_3 = a_0a_1a_2a_3\overline{a_4a_5a_6a_7}$ ..., puis à une distance 8, etc. Ainsi, pour un mot de 8 bits, 3 séquences sont nécessaires, 4 pour un mot de 16 bits, et en règle générale, n pour un mot de 2^n bits.

Il est possible de réduire la taille des séquences en remarquant que la transition D_0 vers D_1 de (7) ou D_0 vers D_2 de (8) n'apporte aucune augmentation du taux de couverture. Ainsi, il est possible de séparer la séquence $D_0, \overline{D_0}, D_0$ des autres séquences. La suite de séquences à appliquer pour la couverture de toutes les fautes de couplage idempotentes pour un mot de 8 bits est donc :

$$D_0, \overline{D_0}, D_0,\ D_1, \overline{D_1}, D_1,\ D_2, \overline{D_2}, D_2,\ D_3, \overline{D_3}, D_3 \qquad (9)$$

Pour la couverture des fautes, on ne s'intéresse qu'aux transitions entre les mots, ce qui se traduit, en terme de lectures/écritures par :

$$\updownarrow(w_{\overline{D_0}}, r_{\overline{D_0}}, w_{D_0}, r_{D_0})$$

$$\updownarrow(w_{D_1}, w_{\overline{D_1}}, r_{\overline{D_1}}, w_{D_1}, r_{D_1})$$

$$\updownarrow(w_{D_2}, w_{\overline{D_2}}, r_{\overline{D_2}}, w_{D_2}, r_{D_2})$$

$$\updownarrow(w_{D_3}, w_{\overline{D_3}}, r_{\overline{D_3}}, w_{D_3}, r_{D_3}, w_{D_0}) \qquad (10)$$

La dernière écriture de (10) sert uniquement à récupérer la donnée D_0 de façon à restituer le contexte en l'absence de faute. Cette écriture est effectuée par transformation logique inverse sur D_3. Nous avons ainsi réalisé un algorithme de test transparent permettant la couverture des fautes de couplage idempotentes intra-mots. Le modèle de fautes de couplage idempotentes est symétrique. Cela nous assure donc que le test transparent (10) conserve le même taux de couverture de fautes que son homologue non transparent.

Il est possible d'étendre cet algorithme pour couvrir les fautes de couplage d'état ou d'influence écriture/lecture. Les diagrammes d'état permettant la sensibilisation de ces fautes de couplages sont représentés figure 4. Les transitions nécessaires pour couvrir toutes les fautes de couplage d'état ne représentent qu'un sous-ensemble des transitions servant à couvrir les fautes de couplage idempotentes. Il est donc facile de déduire l'algorithme qui permet la couverture de telles fautes. Les fautes d'influence peuvent être couvertes, comme le montre la figure 4b, par l'ajout de lectures des données après les transitions, et donc par l'algorithme suivant :

$$\updownarrow(w_{\overline{D_0}}, r_{\overline{D_0}}, r_{\overline{D_0}}, w_{D_0}, r_{D_0}, r_{D_0})$$

$$\updownarrow(w_{D_1}, w_{\overline{D_1}}, r_{\overline{D_1}}, r_{\overline{D_1}}, w_{D_1}, r_{D_1}, r_{D_1})$$

$$\updownarrow(w_{D_2}, w_{\overline{D_2}}, r_{\overline{D_2}}, r_{\overline{D_2}}, w_{D_2}, r_{D_2}, r_{D_2})$$

$$\updownarrow(w_{D_3}, w_{\overline{D_3}}, r_{\overline{D_3}}, r_{\overline{D_3}}, w_{D_3}, r_{D_3}, r_{D_3}, w_{D_0}) \qquad (12)$$

3.2.7 Algorithme général

Il est maintenant possible de déduire l'algorithme général basé sur l'algorithme de MARCH B et qui possède les extensions pour le test transparent des mémoires orientées mots. Celui-ci est maintenant donné pour des mots de 8 bits :

$$\uparrow (r_{D_0}, r_{D_0}, w_{\overline{D_0}}, r_{\overline{D_0}}, r_{\overline{D_0}}, w_{D_0}, r_{D_0}, r_{D_0}, w_{\overline{D_0}})$$

$$\uparrow (r_{\overline{D_0}}, r_{\overline{D_0}}, w_{D_0}, w_{\overline{D_0}})$$

$$\downarrow (r_{\overline{D_0}}, r_{\overline{D_0}}, w_{D_0}, w_{\overline{D_0}}, w_{D_0})$$

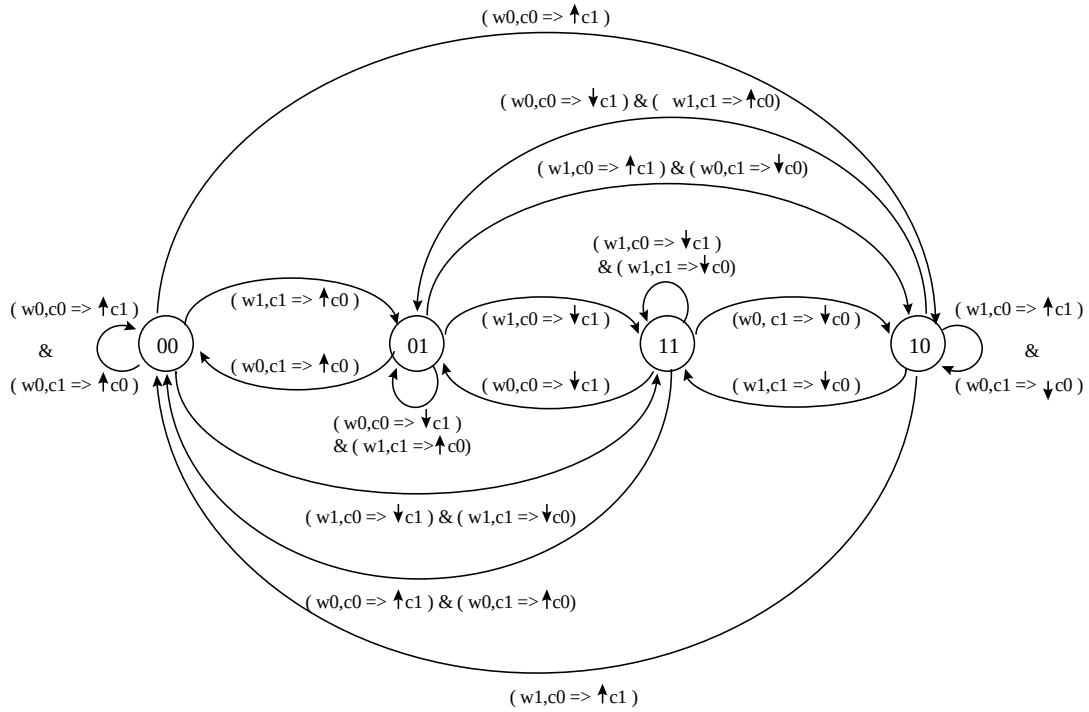
$$\downarrow (r_{D_0}, r_{D_0}, w_{\overline{D_0}}, w_{D_0})$$

$$\uparrow (w_{D_1}, w_{\overline{D_1}}, r_{\overline{D_1}}, r_{\overline{D_1}}, w_{D_1}, r_{D_1}, r_{D_1})$$

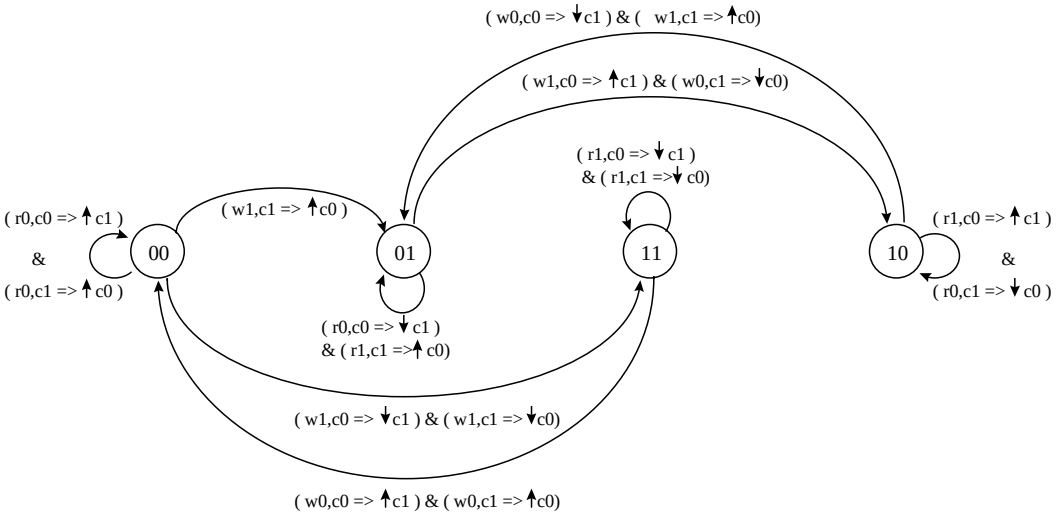
$$\uparrow (w_{D_2}, w_{\overline{D_2}}, r_{\overline{D_2}}, r_{\overline{D_2}}, w_{D_2}, r_{D_2}, r_{D_2})$$

$$\uparrow (w_{D_3}, w_{\overline{D_3}}, r_{\overline{D_3}}, r_{\overline{D_3}}, w_{D_3}, r_{D_3}, r_{D_3}, w_{D_0}) \qquad \textit{(13)}$$

Cet algorithme est très complet puisqu'il permet la couverture de toutes les fautes de cellule simple ainsi que les fautes de couplage idempotentes, d'états et d'influence lecture/écriture inter et intra-mots.



(a)



(b)

Figure 4 : Diagrammes de sensibilisation aux fautes d'influence écriture/lecture et d'état. (a) diagramme de sensibilisation des influences d'écriture. (b) diagramme minimum de sensibilisation des influences écriture/lecture

Par ailleurs, le caractère transparent du test en fait une procédure plus puissante que le test classique. En effet, le test s'effectuant avec des données non déterminées à l'avance, une couverture de certaines fautes non modélisées est réalisée à chaque nouvelle procédure de test.

Il est également important de noter que le temps d'exécution de l'algorithme est diminué par rapport au test (moins efficace) utilisant différentes séquences d'initialisations pour couvrir les fautes intra-mots. Ainsi, pour une mémoire RAM de N mots, la durée de l'algorithme est de :

- cycles pour des mots de 8 bits contre 17*4*N = 68*N cycles d'un test utilisant les 4 séquences d'initialisations (00000000, 01010101, 00110011 et 00001111) ;
- cycles pour des mots de 16-bits contre 17*5*N = 85*N cycles pour un test utilisant 5 séquences d'initialisation ;
- cycles pour des mots de 64-bits contre 17*7*N = 119*N cycles pour un test utilisant 7 séquences d'initialisation ;
- plus généralement, de (23+7*n)*N cycles pour des mots de 2^n bits.

A partir de l'algorithme de test, nous déterminons maintenant le schéma BIST associé au processeur du tableau SIMD.

3.2.8 Schéma BIST

Le test intégré s'effectue en 3 phases : la génération des vecteurs de test, la compression des résultats intermédiaires et la comparaison du résultat compressé avec la signature.

3.2.8.1 Génération des vecteurs de test

Les vecteurs d'un test transparent sont constitués par les données des mémoires ou une forme modifiée de ces données obtenue par des opérations logiques simples (complémentation de certains bits). Afin de ne pas ajouter de multiplexeurs dans les chemins de données, il est naturel de confier ces opérations à l'unité logique du processeur.

Pour des largeurs de chemins de données de 2^n bits, n opérations logiques, correspondant aux n opérations nécessaires pour la procédure de test transparent, sont ajoutées à l'unité logique. Le coût du test est alors réduit : un ajout, précédent l'étape de synthèse, de fonctions simples à la partie arithmétique et logique n'entraîne pas de surcoût important. De même, si on utilise des IP logicielles, les ajouts sont très faibles et n'entraînent pas de changement important, l'insertion de testabilité ne modifiant pas les chemins de données entre blocs.

3.2.8.2 Compression des résultats

La compression consiste, à partir des résultats obtenus par l'application des vecteurs de test, à produire un ou quelques mots représentatifs de ces résultats. Ces quelques mots sont ensuite comparés avec les valeurs fonctionnellement correctes. La difficulté générale de la compression est de ne pas perdre d'information, donc de posséder un faible aliasing. Celui-ci est défini comme la probabilité qu'une réponse fautive donne un résultat correct après compression (voir chapitre 2).

Dans le cas du test transparent, les difficultés au niveau de la compression sont importantes, car les vecteurs de test ne sont pas connus. Toute l'astuce consiste alors à utiliser une compression ayant un faible aliasing et permettant d'obtenir, en cas d'absence de fautes, un résultat connu indépendant des données initiales.

Une solution, proposée dans [Nicolaidis1996] consiste à utiliser un registre à décalage et à rétroaction linéaire (LFSR - Linear Feedback Shift Register). Toutefois, elle implique un surplus matériel non négligeable, et son insertion dans le schéma du processeur ne paraît pas évidente. D'autre part, l'utilisation de LFSR pour la compression du test transparent implique un surplus de temps assez important, puisque le résultat doit être pré-calculé à chaque fois.

Une compression des résultats pour les circuits arithmétiques a été proposée par Rajsky et al. [Rasjky and Tyszer1993] et reprise par Mukherjee et al. [Mukherjee1995, Mukherjee1997] : elle est réalisée grâce à un additionneur avec retenue rotative (rotate-carry). Sa structure est la suivante : la retenue générée par le bit de poids fort est stockée puis ré-injectée au cycle suivant dans la position correspondant au bit de poids faible. Cela permet de diminuer la perte d'information due au dépassement de la capacité de l'additionneur.

[Rajsky and Tyszer1993] ont montré, par l'analyse de cette structure de compression, que l'aliasing obtenu, à la fois pour l'additionneur et le circuit sous test, était de l'ordre de $(2^n-1)^{-1}$ (n étant la largeur du chemin de données). Ces performances sont donc comparables à celles des LFSR et automates cellulaires. Par exemple, pour n=8, l'aliasing est de 0,39 % et pour n=16, il est de 0,0015 %. On voit qu'Il diminue fortement avec l'augmentation de la largeur des mots. Ainsi, pour n=32, il est de $2{,}3.10^{-8}$ %, ce qui est très acceptable. De plus, ce taux d'aliasing est donné pour des fautes dans le circuit sous test et l'additionneur, alors que, comme nous le verrons plus tard, l'additionneur est à nouveau testé par les procédures utilisées pour les chemins de données, ce qui supprime une partie de l'aliasing.

L'utilisation d'un additionneur comme compresseur sera retenue dans notre schéma de test pour les raisons suivantes : d'une part, il satisfait aux exigences de compression et de faible surplus matériel ; d'autre part, il permet de déterminer à l'avance le résultat de la compression du test des mémoires sans connaître les données d'origine et sans aucun pré-calcul. En effet, les séquences de MARCH utilisées se composent de suites de lecture d'une donnée et de l'inverse de la donnée. Or, si $\oplus$ symbolise l'addition avec retenue rotative, et si $a_0, a_1, \ldots a_n$ sont les vecteurs de test successifs, on peut écrire :

$a_0 \oplus \overline{a_0} = 1111\ldots11$,

$a_0 \oplus \overline{a_0} \oplus a_1 \oplus \overline{a_1} = 1111\ldots11$ si $a_1 \neq 000\ldots00$

et $a_0 \oplus \overline{a_0} \oplus a_1 \oplus \overline{a_1} \oplus \ldots \oplus a_n \oplus \overline{a_n} = 1111\ldots11$ si $a_n \neq 000\ldots00$

Et si la signature S est réalisée par l'opération suivante :

$S = a_0 \oplus \overline{a_0} \oplus a_1 \oplus \overline{a_1} \oplus \ldots \oplus a_n \oplus \overline{a_n} \oplus 000\ldots00$,

alors $S = 111..11$ *quelles que soient les données de départ* $a_0, a_1, ... a_n$.

Le résultat de la compression est donc connu à condition que le nombre de lectures de la donnée et de l'inverse de la donnée soit identique. Ceci est vérifié pour l'algorithme proposé. Il est par ailleurs intéressant de noter que l'opération de lecture n'étant pas considérée comme destructive, il est toujours possible d'ajouter une lecture à un algorithme de test mémoire afin de réaliser cette condition.

3.2.8.3 Analyse de signature

L'étape d'analyse de signature découle alors simplement, puisqu'il n'est nécessaire de disposer que d'une seule donnée S = 111...11, qui doit être préchargée dans tous les processeurs et avec laquelle s'effectue la comparaison, soit à la fin de toute la procédure de test, soit, si on désire encore diminuer les problèmes d'aliasing, après chaque étape comportant un nombre identique de lecture de données et de leurs inverses.

Le but du test étant d'obtenir l'état du processeur, et non de déterminer les parties fautives de celui-ci, il est suffisant d'obtenir en fin de test un seul bit représentant l'état général du processeur. Nous avons appelé ce bit le test d'intégrité (t_i). Ce dernier est très important et constitue le point faible du test, puisqu'un simple collage ou une simple faute dans la partie de calcul du test d'intégrité peut entraîner une incapacité à détecter les fautes et donc une mise en échec du test.

Afin de se prémunir contre un tel comportement, deux précautions sont prises. La première consiste à vérifier, avant toute procédure de test, le fonctionnement correct du test d'intégrité (son passage à 0 et 1). La deuxième précaution consiste à doubler la partie opérative servant à la vérification de la signature et donc au calcul du test d'intégrité. Ainsi, la signature S sera comparée à la valeur correcte C de deux façons différentes :

- par leur soustraction, le drapeau Z indiquant alors la mise à 0 du résultat ;
- par un XOR entre C et S.

Ces deux parties, pré-existantes dans la partie opérative nécessitent une précaution lors de leur synthèse, car elles doivent être physiquement séparées. Néanmoins, le surplus matériel est très peu important comme nous le verrons dans la partie résultats.

3.2.9 Conclusion

Nous avons présenté une méthode complète de test de mémoires. Les modifications de la structure sont de trois ordres et ne concernent que l'ALU : ajout d'une retenue rotative à l'additionneur, ajouts de fonctions dans la partie logique de l'opérateur, mise en place du bit de test d'intégrité. Ces modifications sont peu importantes.

Ainsi, les caractéristiques principales du test sont :

- Test transparent, adapté aux mémoires orientées mots, et de durée raisonnable, ce qui autorise l'utilisation de cette procédure pour un test périodique des processeurs ;
- Test sans intrusion dans les chemins de donnée, d'où une perte en performance minimale ;

- Test de March B et compression des résultats de test avec un aliasing minimal, d'où une très bonne couverture de fautes.

Ce test est également utilisé pour le test du tableau de registres, et pourrait être utilisé pour tout bloc de mémorisation relié aux chemins de donnée comme des tables de correspondance mémoire (TLB : Translation Lookehead Buffer).

La partie mémoire étant testée, l'étape suivante concerne le test de la partie opérative, ou plus généralement, du chemin de données.

3.3 Test du chemin de données

3.3.1 Introduction

Le chemin de données est constitué de l'ALU, du multiplieur, ainsi que des multiplexeurs se trouvant sur le chemin. Dans cette section, les phases de génération de vecteurs de test et de compression des résultats adaptés à celui-ci sont présentées. Mais il est tout d'abord nécessaire de déterminer le modèle de fautes que nous allons chercher à détecter.

3.3.2 Le modèle de fautes

Le modèle de fautes de collage, qui est encore actuellement le modèle de fautes le plus utilisé dans l'industrie, ne représente pas convenablement les défauts des circuits actuels (chapitre 2). Ce modèle de fautes n'est tout de même pas à rejeter directement car, d'une part, il sert de référence, et d'autre part, la couverture des fautes de collage entraîne automatiquement la couverture d'un nombre important d'autres fautes. Toutefois, il est inadapté au problème que nous nous sommes posés, car il suppose connue la structure de chaque bloc. Pour la même raison, les modèles de fautes structurelles ne peuvent pas être retenus.

Un modèle de fautes fonctionnel intéressant est le modèle de fautes de cellule (CFM : Cell Fault Model). Il est principalement utilisé pour les « tableaux de logique itérative » (ILA : Iterative Logic Array, figure 5). Les ILA sont des structures régulières formées d'ensembles de cellules identiques. Ce modèle est donc particulièrement bien adapté aux opérateurs logiques constitués, pour la plupart, de nombreux blocs identiques formant des ILA mono ou bi-dimensionnels. On peut par ailleurs montrer que ce modèle inclut celui de collage, ce qui lui donne une certaine légitimité.

Le modèle CFM se définit ainsi, en supposant qu'une seule cellule est fautive à un instant donné : une faute CFM est toute faute survenant dans la cellule qui change la fonction combinatoire de celle-ci. Ce modèle exclut donc les fautes qui ont un comportement séquentiel, ce qui constitue sa principale faiblesse.

Pour couvrir l'ensemble des fautes du modèle CFM, il est alors nécessaire d'appliquer toutes les entrées possibles à chaque cellule élémentaire de l'ILA. Cela n’est pas toujours possible pour toutes les cellules. La notion de testabilité est alors utilisée afin de mesurer le pourcentage de couverture du modèle de fautes CFM qui peut être atteint sur une structure.

La testabilité des ILA et de quelques opérateurs usuels par rapport à ce modèle de fautes va tout d'abord être rappelée. Puis nous présentons le schéma BIST pour le test du chemin de données du processeur.

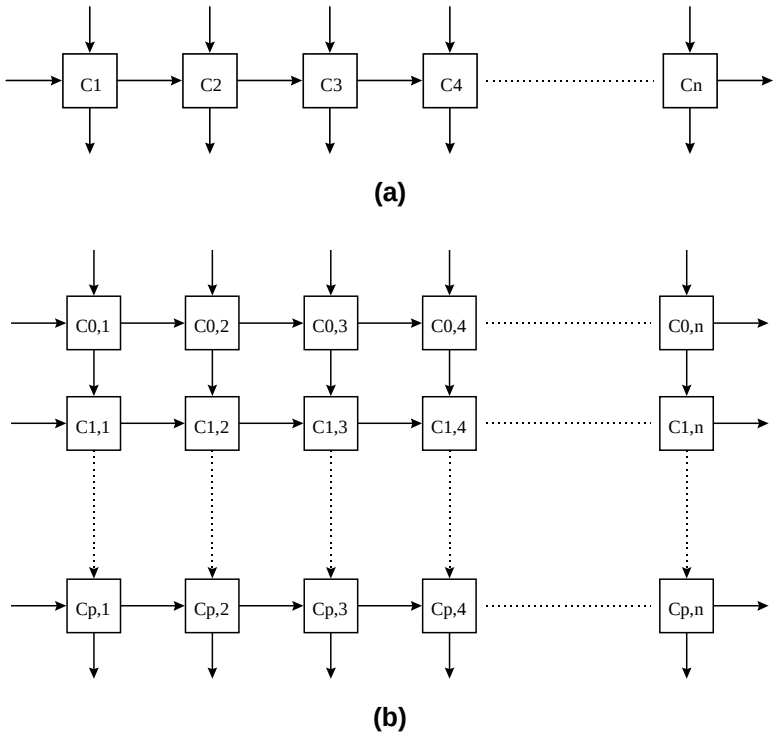


Figure 5 : iterative Logic Arrays (ILA) (a) en une dimension, (b) en deux dimensions

3.3.3 Testabilité des ILA et de quelques opérateurs usuels

Par rapport au modèle de faute CFM, la testabilité d'un opérateur logique dépend de :

1. La possibilité de pouvoir appliquer toutes les entrées possibles à chaque cellule de l'ILA (observabilité) ;
2. La probabilité qu'une sortie fautive d'une cellule se propage jusqu'à une sortie primaire (contrôlabilité).

Afin de mesurer la contrôlabilité et l'oservabilité, on définit pour chaque cellule i de l'ILA la testabilité $T_i = \frac{V_i}{2^k} * 100\%$, V_i étant le nombre de vecteurs de test différents appliqués aux entrées de la cellule et k le nombre d'entrées de la cellule i.

Lorsque l'ILA comprend N cellules de même type, chacune d'elle ayant une testabilité T_i, la testabilité de l'ILA est : $T_{ILA} = \frac{\sum_{i=1}^{N} T_i}{N}$

De façon plus générale, lorsque l'ILA est constitué de q cellules de types différents, chacune d'entre elles possédant k_j entrées, une testabilité T_j et étant présentes N_j fois dans le bloc, la testabilité de l'ILA devient :

$$T_{ILA} = \frac{\sum_{j=1}^{q} N_j 2^{k_j} T_j}{\sum_{j=1}^{q} N_j 2^{k_j}}$$

Il est alors possible de déduire de cette formule la testabilité de quelques opérateurs usuels :

- pour un additionneur composé d'additionneurs complets avec propagation de retenue (ILA mono-dimensionnel), la testabilité est de 100 % ;
- de même, la testabilité d'un soustracteur obtenu par propagation de retenue sur des soustracteurs complets est de 100 % ;
- la testabilité de multiplieurs est plus difficile à obtenir. Toutefois, dans [Gizopoulos1996], la testabilité de 3 multiplieurs est donnée : entre 99,44 % et 99,87 % pour un multiplieur à propagation de retenue, entre 99,48 % et 99,87 % pour un multiplieur à sauvegarde de retenue, et entre 99,71 % et 99,91 % pour un multiplieur de Booth.

Ces quelques exemples d'opérateurs usuels et de bonne régularité ne permettent pas de déduire la testabilité pour des opérateurs plus complexes, plus rapides et présentant une moins bonne régularité. Il est clair que la testabilité baisse en même temps que la régularité de la structure diminue. Toutefois, la plupart des opérateurs utilisés usuellement sont formés de structures ayant une bonne régularité, ceci afin de réduire leur encombrement, et donc d'augmenter leur performance. Par conséquent, on peut espérer obtenir un bon taux de couverture des fautes CFM, même pour des structures de calcul très rapides.

3.3.4 Schéma BIST

Le problème le plus difficile à résoudre concerne le choix des vecteurs de test adaptés au modèle de fautes CFM, ainsi que leur élaboration.

Ce point a été étudié dans [Mukherjee1995] qui montre que les vecteurs de test adaptés aux structures régulières des ILA sont des données exhaustives par blocs de 4 bits. Un exemple de table de vérité pour les 4 bits de poids faible d'un additionneur à propagation de retenue est donné dans le tableau 2. On peut vérifier dans celui-ci que des données exhaustives de 4 bits permettent de produire toutes les entrées des additionneurs complets (cases entourées en foncé). On peut également vérifier que toute erreur unique dans les cellules se propage jusqu'à une sortie primaire. cela signifie que cet additionneur a un taux de couverture des fautes CFM de 100 % avec seulement 16*16, soit 256 vecteurs de test. On peut également déduire du tableau 2 que les vecteurs nuls sont inutiles, et peuvent donc être supprimés pour arriver à un test comprenant uniquement 225 vecteurs de test. Cette propriété peut être étendue à toutes les structures de type ILA. Ainsi, le nombre très faible de vecteurs de test est adapté au critère de test périodique.

a_0	b_0	S_0	C_1	a_1	b_1	S_1	C_2	a_2	b_2	S_2	C_3	a_3	b_3
0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	1	1	0	0	0	0	0	0	1
0	0	0	0	1	0	1	0	0	0	0	0	1	0
0	0	0	0	1	1	0	1	0	0	1	0	1	1
0	1	1	0	0	0	0	0	0	1	1	0	0	0
0	1	1	0	0	1	1	0	0	1	1	0	0	1
0	1	1	0	1	0	1	0	0	1	1	0	1	0
0	1	1	0	1	1	0	1	0	1	0	1	1	1
1	0	1	0	0	0	0	0	1	0	1	0	0	0
1	0	1	0	0	1	1	0	1	0	1	0	0	1
1	0	1	0	1	0	1	0	1	0	1	0	1	0
1	0	1	0	1	1	0	1	1	0	0	1	1	1
1	1	0	1	0	0	1	0	1	1	0	1	0	0
1	1	0	1	0	1	0	1	1	1	1	1	0	1
1	1	0	1	1	0	0	1	1	1	1	1	1	0
1	1	0	1	1	1	1	1	1	1	1	1	1	1

Tableau 2 : table de vérité des 4 premiers additionneurs complets d'un additionneur à propagation de retenue. Les cases en foncé indiquent les configurations différentes obtenues en entrées de chacun des additionneurs complets

Dans notre cas, l'additionneur peut être réalisé de deux façons différentes : par la bibliothèque du fabricant de circuit intégré ou par une programmation VHDL structurelle suivie de synthèse. Dans le premier cas, nous ne connaissons pas la structure de l'opérateur. Dans le second cas, cette structure est connue et il est alors possible de calculer de façon précise le taux de couverture obtenu par la méthode. Afin de prendre en compte ces deux cas, nous ne ferons pas d'hypothèse sur la structure interne des opérateurs, et le calcul de la couverture de fautes sera fait uniquement à partir du modèle de collage pour lequel nous disposons d'un simulateur (QuickFault2 de Mentor Graphics). Nous vérifierons toutefois que la méthode est adaptée à différents niveaux d'optimisation (et donc à différentes structures).

Les opérateurs logiques et de décalage, qui possèdent également une structure très régulière, ne présentent pas de problème particulier de testabilité. Le multiplieur est une structure un peu moins

régulière, et souvent plus délicate à tester en raison de la troncature des bits de poids faible qui limite fortement l'observabilité des fautes. Dans notre cas, ce problème n'existe pas puisque la troncature du multiplieur s'effectue au niveau des opérandes. Le lecteur intéressé par les problèmes de test et de schéma BIST des multiplieurs à troncature de bits de poids faibles peut se référer à [Mukherjee1997].

Afin de ne pas pénaliser le chemin de données, nous avons choisi de réaliser la génération des vecteurs de test pseudo-exhaustifs par deux registres transformés en LFSR. Les LFSR utilisent la division polynomiale [Agrawal1993, Mc Cluskey1985] pour produire des données à partir des données précédentes. Ils permettent ainsi d'obtenir des vecteurs de test pseudo-aléatoires ou pseudo-exhaustifs.

Leur structure générale est représentée figure 6a. La division polynomiale offre un très grand nombre de possibilités dont celle de pouvoir déterminer une séquence incluant des vecteurs de test déterminés [Dufaza and Zorian1997]. La configuration que nous utiliserons est représentée figure 6b. Elle permet de produire tous les nombres de 4 bits une et une seule fois par cycle, à l'exception du 0, inutile pour la couverture des fautes CFM. On obtient ainsi un générateur de vecteurs exhaustifs par blocs de 4 bits, appelé également générateur exhaustif.

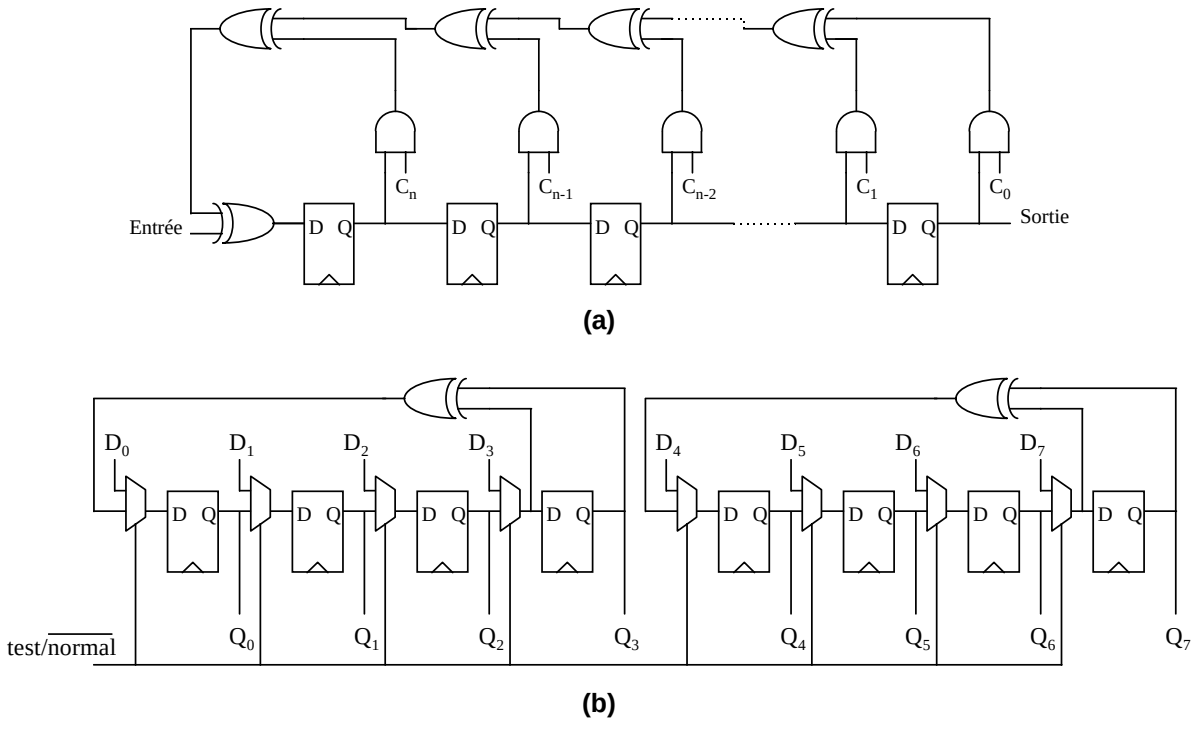


Figure 6 : Linear Feedback Shift Register (LFSR) (a) forme générale de type « externe » (b) registre 8 bits avec fonctionnement possible en générateur de vecteurs de test exhaustifs sur chaque bloc de 4 bits. La fonction de division polynomiale est x^4+x^3+1. Le signal test commande le basculement entre les deux fonctionnements

Deux registres du tableau de registre sont alors modifiés de façon à autoriser deux fonctionnements : un fonctionnement normal correspondant à celui de registres de données classiques, et un fonctionnement de test correspondant à un fonctionnement en registres pseudo-exhaustifs générateurs de vecteurs de test.

La compression des données s'effectue grâce à l'additionneur muni de retenue rotative présenté dans la section précédente. L'obtention de la signature s'effectue également de manière similaire à celle indiquée dans la section précédente. Le résultat des deux tests est donc synthétisé par le drapeau de test d'intégrité qui indique l'état global du processeur.

Le surplus matériel de cette méthode de test est uniquement lié à la modification de deux registres afin qu'ils produisent les vecteurs de test pseudo-exhautifs (figure 6). Les modifications de la description sont donc minimes et peu pénalisantes, puisque le chemin de données n'est pas modifié.

3.4 Test « hors-ligne » des autres éléments : quelques pistes

3.4.1 Introduction

Nous avons vu, dans les sections précédentes, le test BIST de la mémoire et des chemins de données. Ces deux blocs forment la majeure partie des processeurs utilisés dans les tableaux SIMD. Selon la complexité du processeur, d'autres blocs sont toutefois nécessaires ou peuvent être ajoutés. Parmi ces blocs, le décodeur est toujours présent. Il permet le décodage des micro-instructions vers les blocs fonctionnels internes.

Un contrôleur peut également être ajouté au schéma général. Il sert alors à séquencer les micro-instructions, ou, plus probablement dans le cas les processeurs de calculateurs SIMD, à contrôler le réseau. Il permet ainsi de soulager le processeur des problèmes de gestion du réseau. Le contrôleur est alors une simple machine d'état, purement séquentielle.

On peut également imaginer d'autres évolutions du processeur vers une version se rapprochant plus des processeurs complexes de type RISC. Toutefois l'avantage des processeurs SIMD réside dans leur simplicité qui leur permet de traiter des données en un seul cycle d'horloge. Nous supposerons donc que seuls le contrôleur et le décodeur peuvent être ajoutés à la mémoire et aux chemins de données.

3.4.2 Le décodeur

Le décodeur permet de faire la liaison entre les micro-instructions et les blocs internes du processeur. Dans notre cas, aucun test spécifique ne lui est associé : de même que le décodage des adresses mémoires ne nécessite aucun test particulier, le test du décodage des micro-instructions, dans notre cas, est considéré comme inutile. En effet, toutes les phases de test des mémoires et du chemin de données sont contrôlées depuis l'extérieur du processeur via les micro-instructions. Pour un test complet de chaque partie du processeur, toutes les parties du décodage sont utilisées avec les combinaisons utiles du décodage. Par conséquent, on réalise un test fonctionnel total du bloc de décodage qui est jugé comme suffisant.

3.4.3 Le contrôleur

Nous supposerons que le contrôleur a un comportement purement séquentiel de type machine d'état. Celui-ci est très difficile à tester, car le fait même du comportement séquentiel augmente de façon considérable la complexité des procédures de test. Ainsi, le test devient un problème NP-complet.

Par conséquent, on utilise habituellement une solution à base de scan path : les bascules du contrôleur sont réunies sous forme de chaînes de scan (voir chapitre 2). Les bascules considérées comme entrées du circuit sont alors organisées en LFSR pseudo-aléatoires, alors que les autres bascules servent à la compression des données. L'inconvénient majeur d'une telle approche est la profonde modification de la structure qui peut entraîner une perte en performance importante de ce bloc. De plus, les parties de la logique qui ne sont pas situées entre deux bascules ne sont pas testées. Cette solution nous paraît cependant la seule viable actuellement.

3.4.4 Conclusion

Deux éléments gravitent autour du chemin de données et de la mémoire : il s'agit du décodeur et du contrôleur. Dans notre cas, le décodeur est testé de façon fonctionnelle par les procédures de test des autres éléments. Quant au contrôleur, il peut être testé de manière totalement indépendante au reste du circuit par des techniques de BIST classiques constituées d'un générateur de vecteurs de test pseudo-aléatoires par LFSR et d'une compression des données par LFSR également. La méthode de test de ces structures qui s'intégrerait parfaitement à celle de test périodique reste néanmoins à étudier de façon précise.

3.5 Résultats

3.5.1 Introduction

Afin de valider les méthodes de test, nous avons conçu un processeur en langage de description matérielle VHDL synthétisable. La largeur du chemin de données pour ce prototype est de 16 bits et le processeur est composé de 6 blocs correspondant à la description de la figure 2 (paragraphe 3.1) :

- un décodeur des micro-instructions ;
- un bloc de registres partagé en deux tableaux de 8 registres ;
- une mémoire RAM de 1 K-mots ;
- un bloc d'interconnexions permettant de faire la liaison entre les mémoires, l'extérieur du processeur et les parties opératives ;
- une ALU comprenant des fonctions logiques (aandb, aorb, axorb, nota, notb ainsi que 3 fonctions de test mémoire), arithmétiques (aaddb, asousb, idema, idemb, C2a), et de décalage (slaa, slla, srla). Cette ALU comprend également un générateur de drapeaux d'overflow V, de carry C, de comparaison (= et >) Z et GT, de signe S et de parité P ;
- un multiplieur 8*8 avec résultat sur 16 bits.

Ce modèle a ensuite été synthétisé et optimisé à l'aide de l'outil autologic de MENTOR, à l'exception de la mémoire RAM qui n'a pu être synthétisée en raison de l'absence d'outils de synthèse de mémoire. Une librairie MHS est utilisée comme librairie de synthèse. Elle est caractérisée par une largeur de grille du transistor de 0,6 μm. Les multiplieurs et additionneurs sont ceux proposés par l'outil de synthèse. Aucune hypothèse n'est donc faite sur leur structure qui diffère selon le niveau d'optimisation demandé. Les contraintes de temps de traversée des additionneurs et du multiplieur sont fortes.

3.5.2 Couverture des fautes de collage

Nous avons ensuite utilisé l'outil Quick-Fault2 de MENTOR afin de produire les fautes de collage sur le schéma obtenu et de juger de l'efficacité de notre méthode de test intégré sur ce modèle de fautes. Les résultats obtenus sont les suivants :

	décodeur	registres	interconnexions	UAL+drapeaux	multiplieur	total
fautes créées	330	5628	894	1850	1088	9940
fautes testables	329	5628	894	1772	1088	9861
fautes détectées	281	5603	750	1677	1088	9549
fautes détectées / fautes créées %	85,15 %	99,56 %	83,89 %	90,65 %	100 %	96,07 %

Tableau 3 : résultats de couverture des fautes de collage sur les blocs du processeur

Les carences en test sont de deux ordres : la plus importante concerne les connexions avec l'extérieur qui ne sont pas testées par la méthode. Cela entraîne une baisse du taux de couverture au niveau du bloc de connexions internes, mais également du bloc de décodage pour lequel on ne teste pas les configurations prenant en compte les connexions externes. Cette carence est supprimée par la suite grâce au test des interconnexions entre processeurs. La deuxième carence concerne les drapeaux de l'ALU, ces drapeaux n'étant pas observables. Un test d'intégrité dont le résultat serait basé sur la valeur de tous les drapeaux pourrait palier ce problème.

De façon générale, on peut confirmer que la procédure de test permet d'obtenir de bons résultats de couverture des fautes de collage, ce qui permet d'assurer une certaine qualité au test. En effet, la couverture de fautes est équivalente à celle obtenue par scan path pour un nombre réduit de vecteurs de test.

Un des objectifs que nous nous sommes fixés est de pouvoir utiliser la méthode de test sur des structures similaires à celle que nous venons de présenter. L'utilisation de tests fonctionnels est ainsi liée à cet objectif. Afin de vérifier que celui-ci est atteint, nous avons synthétisé différents additionneurs au niveau structurel. Ces additionneurs sont :

- à propagation de retenue ;
- à retenue anticipée ;
- à retenue prédite, avec différents schémas de prédiction (blocs de 4 ou de 8 bits).

Ces additionneurs présentent des compromis temps de traversée / surface différents. Toutefois, pour chacun de ces schémas, un taux de couverture de 100 % des fautes de collage a été obtenu avec les vecteurs de test proposés.

De même, nous avons vérifié que le multiplieur proposé dans la synthèse était un multiplieur de Booth. Un essai a donc été fait avec un multiplieur à structure moins régulière comme le multiplieur de Wallace.

Pour un multiplieur de Wallace 8*8 => 16, le résultat est donné dans le tableau 4.

fautes créées	fautes testables	fautes détectées	fautes détectées / fautes créées %
1374	1366	1363	99,2 %

Tableau 4 : couverture des fautes de collage pour un multiplieur de Wallace 8*8

Les fautes de collage ne sont donc pas entièrement testées. Le résultat obtenu est toutefois très correct, puisque seules 3 fautes testables n'ont pas été détectées par la procédure de test, en seulement 225 vecteurs de test.

3.5.3 Autres résultats

Les autres résultats concernent le surplus matériel, la perte en performance, et la durée du test. Le surplus matériel a été calculé par comparaison du nombre de portes équivalentes entre le processeur conçu avec test intégré, et le même processeur sans test intégré. Le nombre de portes équivalentes pour la mémoire a ensuite été estimé, de façon à la prendre en compte dans le calcul du surplus matériel. Ce dernier est évalué pour deux chemins de données de 16 et de 64 bits. Les résultats sont donnés dans le tableau 5.

De ce tableau, on peut lire que le surplus matériel engendré par le test intégré diminue lorsque la largeur du chemin de données augmente. Par ailleurs, ce surplus matériel est très faible lorsqu'on prend en compte la mémoire : il est alors compris entre 0,65 % et 0,8 %.

Le surplus en performance est très faible, puisque aucun multiplexeur n'a été ajouté dans les chemins de donnée. Il est donc limité à une légère augmentation de la longueur de certaines interconnexions due à l'augmentation de surface. Cette faible perte en performance est très difficilement mesurable, car elle dépend du placement et routage du circuit.

	avec test intégré (portes équivalentes)	sans test intégré (portes équivalentes)	surplus matériel (pourcentage)
processeur 16 bits sans mémoire	4535	4237	7,03 %
processeur 16 bits avec mémoire	37303	37005	0,8 %
processeur 64 bits sans mémoire	23983	22974	4,39 %
processeur 64 bits avec mémoire	155055	154046	0,65 %

Tableau 5 : surplus matériel de la technique de BIST employée

Le troisième résultat concerne la durée de l'algorithme de test. Celle-ci est très importante puisque le test hors ligne doit être utilisé en test périodique. Nous avons déjà les éléments de réponse pour la durée du test de mémoire qui est de (23+7*n)*N cycles pour N mots de 2^n bits. Cela représente 51*N cycles pour des mots de 16 bits et 65*N cycles pour des mots de 64 bits. D'autre part, le test d'une fonction arithmétique, logique ou du multiplieur nécessite 225 vecteurs de test, alors que le test des opérations portant sur un seul opérande (NOT, complément à 2, opérations de décalage...) ne nécessite que 15 vecteurs de test.

Ainsi, pour le processeur 16 bits proposé, nous arrivons à un total de (51*(1024+16))+225*6+15*11) = 54555 cycles, soit 0,27 ms pour un temps de cycle de 5 ns correspondant à une fréquence d'horloge de 200 MHz. Pour le même processeur avec une largeur de chemins de 64 bits, on obtiendrait 69115 cycles, soit 0,35 ms à la fréquence d'horloge de 200 MHz. Ces chiffres sont à comparer avec les quelques secondes de test qu'il faudrait si une sauvegarde de contexte était nécessaire. Ce temps peut être légèrement augmenté si des fonctions supplémentaires sont ajoutées. Mais cette augmentation serait alors peu importante car la majeure partie du temps de test est pour la mémoire. Ce temps de test est donc assez faible et compatible avec beaucoup d'applications comme la plupart de celles de traitement d'images. Si la durée de l'algorithme de test est toutefois estimée comme étant trop importante, il est possible de la réduire en limitant le test des mémoires, au détriment de la couverture de fautes.

3.5.4 Conclusion

Une méthode efficace de test a été développée, adaptée aux processeurs simples des tableaux SIMD réalisé directement par VHDL ou à partir d'IP logicielles. Afin de ne pas limiter la méthode au seul processeur présenté dans cette section, les algorithmes développés ont été rendus, autant que possible, indépendants de la structure des blocs logiques. Ceci est vrai aussi bien pour les mémoires (pas d'hypothèse sur la mémoire utilisée) que pour les blocs du chemin de données.

Ainsi, la méthode peut intégrer sans problème d'autres opérateurs au niveau de l'ALU. De même, aucune hypothèse n'a été faite sur la structure des additionneurs et multiplieurs, en préférant une approche plus fonctionnelle à une approche structurelle. Aussi, ces différents blocs peuvent parfaitement être issus d'autres circuits sous forme de blocs matériels optimisés.

Le surplus matériel, les très faibles pertes en performance, la faible durée de l'algorithme ainsi que le bon taux de couverture de fautes font que le test proposé est pleinement efficace pour la structure considérée vis-à-vis des contraintes imposées.

Toutefois, certains points limitent l'approche. Tout d'abord, des éléments de mémorisation qui ne seraient pas connectés au chemin de données ne peuvent pas être testés. Ensuite, comme nous l'avons déjà signalé, le test d'une partie séquentielle s'intègre mal à la méthode. Enfin, le test ne détecte pas les fautes transitoires qui peuvent alors entraîner des résultats erronés sans détection d'erreur au cours du fonctionnement de la structure. Ces fautes sont beaucoup plus nombreuses que les fautes permanentes (on estime en moyenne que 10 fautes transitoires se produisent pour une faute permanente), mais leur effet est moins néfaste sur le système, puisqu'il continue à opérer normalement une fois la faute résorbée. Les fautes transitoires n'influent donc pas sur la durée de vie de la structure, mais sur la probabilité que le résultat d'un calcul soit correct.

Dans l'introduction de ce chapitre, nous avons vu que le test en-ligne sous forme périodique et concurrente (pendant le fonctionnement normal) doit être traité par la méthode proposée lorsque les besoins en fiabilité le nécessitaient. Nous allons donc aborder, dans la section suivante, les problèmes de fautes transitoires et les possibilités de test pour ces fautes.

4 Test « en-ligne » d'un processeur

4.1 Introduction

Il est important de tester les fautes transitoires lorsque l'intégrité du résultat est critique. Deux types d'applications sont concernés : les systèmes critiques type contrôle de trafic, avionique,..., où une faute, à un moment donné, peut entraîner des conséquences dramatiques. Le second type d'application est lié aux problèmes calculatoires de longue durée où une erreur de calcul peut entraîner un résultat complètement faux par effet boule de neige, et fait perdre de longues heures de calcul. Pour ces deux cas, il est important de s'apercevoir qu'une erreur a eu lieu et il peut être utile, soit de pouvoir la corriger, soit de pouvoir retrouver un état correct du système.

Les technologies évoluant vers des niveaux de plus en plus fins, les circuits deviennent de plus en plus sensibles aux événements extérieurs. Par exemple, un événement isolé de type SEU qui, sur les technologies anciennes, ne pouvait guère se produire que dans l'espace, devient une nouvelle cause d'apparition de fautes sur des circuits terrestres. On ne peut donc ignorer ces fautes transitoires.

4.2 Test concurrent

Le test concurrent implique un contrôle continu des données. Dans la suite, nous nous limitons au test en ligne des blocs mémoires et des opérateurs. Par ailleurs, nous supposons qu'une faute au

niveau des instructions entraîne une erreur grave. Elle est donc détectée par des mécanismes logiciels comme les chiens de garde ou les points de rencontre ou alors, entraîne des résultats de test incohérents lors du test périodique.

Les solutions permettant de détecter, voire de corriger, les fautes transitoires des données impliquent une redondance d'informations ou de temps. Cette redondance peut être obtenue par duplication matérielle, par duplication logicielle de code ou de morceaux de code, ou par codage des informations. Nous avons choisi cette dernière solution pour les raisons évoquées dans le chapitre 2.

Le codage de l'information est lié au principe de l'auto-contrôle et est basé sur l'utilisation de codes détecteurs ou correcteurs d'erreurs. Pour satisfaire nos contraintes, nous devons choisir un code qui minimise les pertes en performance et le surplus matériel. Les codes arithmétiques (voir chapitre 2, section 4.7) peuvent être appliqués aux parties arithmétiques ainsi qu'au multiplieur, et minimisent alors le surplus matériel et les baisses en performance dus au test concurrent dans ces structures.

Malheureusement, ces codes sont assez difficiles à coder et décoder. Surtout, ils n'est pas évident de pouvoir les mettre en place afin de détecter des fautes dans les interconnexions et les mémoires où leur coût trop important devient un obstacle à leur utilisation. Enfin, ils ne sont pas conservés dans les opérations logiques et de décalage. Leur mise en place est donc lourde et nécessite, soit un codage et un décodage à l'entrée et à la sortie des blocs arithmétiques, soit de complexes circuits de prédiction pour les autres blocs.

Par opposition, le code de parité est très intéressant pour le test de fautes transitoires apparaissant dans les interconnexions et les mémoires. Il est par ailleurs séparable et très peu coûteux en matériel pour les interconnexions et mémoires. Par contre, les opérations arithmétiques ne conservent pas le code de parité. Elles nécessitent donc une structure de prédiction du bit de parité.

Il est nécessaire de conserver, autant que possible, la compatibilité avec l'unité de contrôle. Or, comme nous l'avons vu, les processeurs séquentiels intègrent souvent des mémoires avec bit de parité. Par ailleurs, il est nécessaire de conserver la cohérence du test pour tout le processeur. Pour ces deux raisons, nous avons choisi la solution basée sur le code de parité pour le test des fautes transitoires. La structure du processeur modifié est présentée figure 7 et les différents blocs la constituant sont présentés au paragraphe suivant.

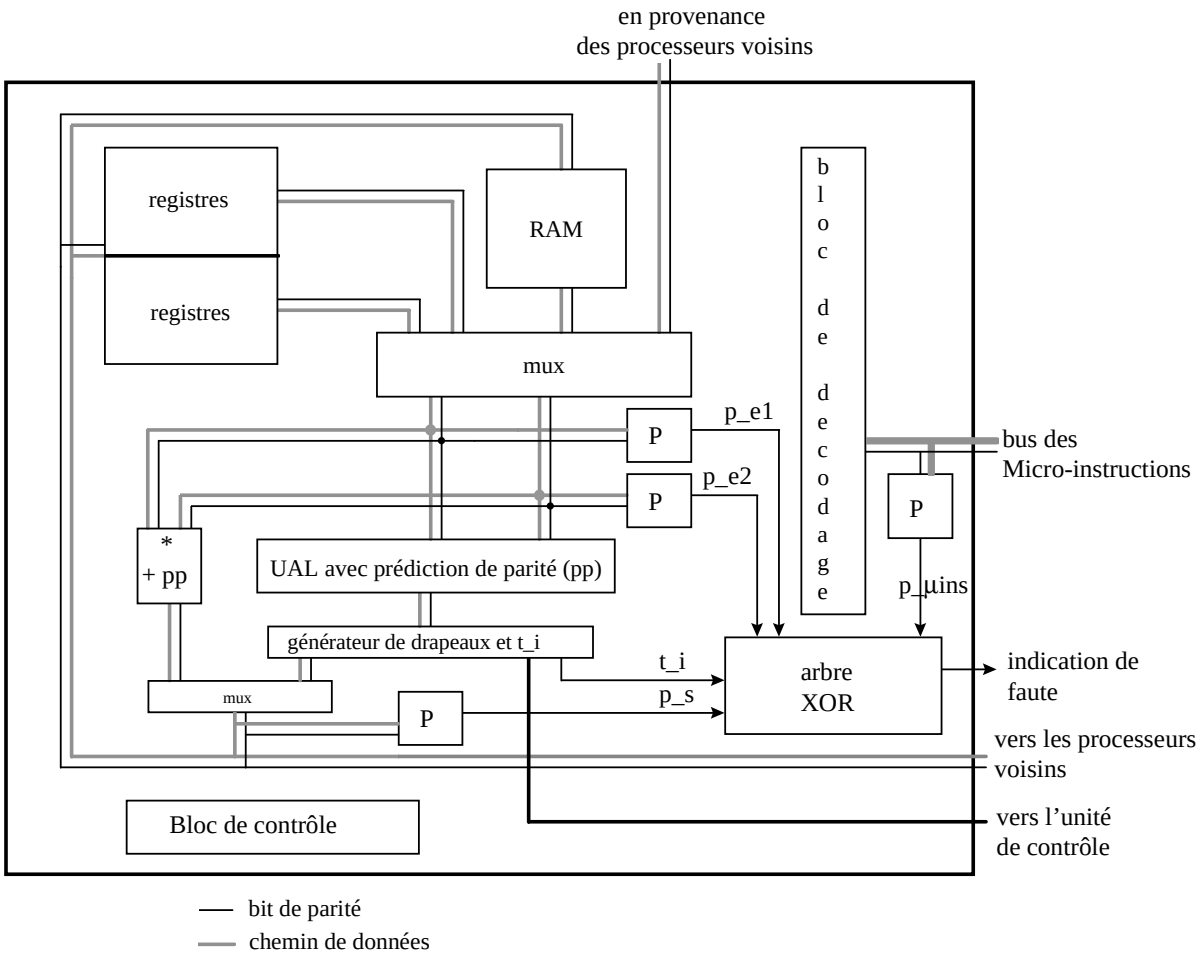


Figure 7 : schéma structurel du processeur équipé de test en-ligne par bit de parité. Sont contrôlées : les micro-instructions entrant dans le circuit (p_µins), les données entrant dans la partie opérative, quelle que soit leur provenance (p_e1, p_e2) et les données sortant de la partie opérative (p_s). Ces 4 bits de parité sont associés au bit de test d'intégrité du test hors-ligne périodique par un arbre de portes ou exclusif fault-secure

4.3 Test en-ligne avec bit de parité

Le test concurrent nécessite l'utilisation de contrôleurs de parité qui doivent être soit totalement auto-contrôlable (TSC), soit fortement code-disjoint (SCD), ceci afin d'assurer la détection d'une faute même lorsque le contrôleur est lui-même fautif (voir chapitre 2, section 4.7). Il est également nécessaire de modifier les opérateurs logiques afin qu'ils prédisent la parité : la parité prédite doit être conçue de manière qu'une faute dans l'opérateur logique soit toujours détectée. C'est la propriété de sûreté par rapport à la faute (fault-secure) qui doit alors être respectée. Ces deux éléments sont maintenant brièvement décrits.

4.3.1 Circuits avec prédiction de parité

Il est assez facile de prédire la parité d'un additionneur, de façon à ce que, en l'absence de fautes, la parité prédite corresponde à la parité du résultat de l'addition. Il est par contre beaucoup plus délicat de prédire la parité de telle façon qu'en présence d'une faute dans le circuit ou dans le mécanisme de prédiction, la parité prédite ne soit jamais en accord avec la parité du résultat. Des travaux ont été

menés dans ce sens, aussi bien pour les opérateurs de décalage [Duarte1997], que pour les opérateurs arithmétiques [Nicolaidis1997]. Quelques résultats sont rappelés ici :

- Pour un circuit de décalage à n positions, le surplus matériel est de 18 % avec une perte en performance de 2,8 %. Le schéma de test est alors composé : d'un décodeur pour l'implémentation de la prédiction de parité ; d'un contrôleur double-rail pour les signaux de contrôle du circuit ; d'un contrôleur de parité basé sur un arbre de portes XOR et sur lequel nous reviendrons dans le paragraphe suivant.
- Pour les circuits arithmétiques, la construction d'un schéma « fault-secure » impose une modification des cellules de base. Celles-ci sont de 2 types : les demi-additionneurs et les additionneurs complets. Elles sont alors remplacées par des cellules incluant une retenue de prédiction Cp qui sert à la prédiction globale de parité. Lorsqu'une sortie d'une de ces cellules n'alimente qu'une seule autre cellule, on peut alors prouver que le circuit obtenu est fault-secure (cas, par exemple, d'un additionneur à propagation de retenue, mais également du multiplieur de Braun). Lorsque le nombre de cellules alimentées est impair, le circuit reste fault-secure, sauf par rapport aux fautes d'interconnexions. Par contre si un nombre de cellules pair est alimenté par une sortie d'une cellule de base, cela entraîne des difficultés importantes pour assurer la propriété fault-secure qui se traduit alors par une augmentation considérable de la surface. Quelques résultats sur différentes structures sont les suivants [Nicolaidis1993], [Nicolaidis1997] :

structure	surplus en surface	baisse en performance
additionneur à retenue anticipée	21,4 %	négligeable
multiplieur de Braun (cas 1)	78,2 à 80,6 %	10,1 à 6,1 %
multiplieur de Braun (cas 2)	49,7 à 47,1 %	22,8 à 18,1 %
multiplieur de Wallace (cas 1)	64,6 à 76,6 %	24,5 à 8,9 %
multiplieur de Wallace (cas 2)	45,8 à 46,7 %	29,2 à 20,6 %

Tableau 6 : résultats en surplus matériel et en baisse en performance de différentes structures d'un additionneur 16 bits et de multiplieurs 8*8 à 32*32

La parité est conservée pour les mémoires et les interconnexions. Les structures de prédiction de parité nous permettent donc d'obtenir une structure globale respectant le schéma de parité et permettant une bonne couverture des fautes transitoires.

4.3.2 Circuits de contrôle de parité

Ces circuits sont conçus pour la plupart à partir de portes XOR deux entrées [Fujiwara and Matsuoka1987, Nikolos1989], même si l'utilisation de bascules et de LFSR a été proposée [Tarnik1995]. [Khakbaz and McCluskey1984] propose une méthode pour construire ces contrôleurs de façon à ce qu'ils soient auto-testables, et [Nikolos1998] améliore cette méthode en proposant un

algorithme général permettant d'obtenir des réalisations optimales de contrôleurs de parité. L'optimalité peut être obtenue selon un des trois critères suivants : nombre minimal de portes (minimisation du surplus matériel), nombre minimal de niveaux (minimisation du temps de traversée du contrôleur), compromis idéal entre ces deux premières contraintes. Nous choisissons de privilégier le temps de traversée des contrôleurs.

La détection d'une faute transitoire est alors certaine. Il faut toutefois s'assurer que le contrôleur reçoit bien un certain nombre de vecteurs de test en entrée. Ce problème est résolu, dans notre cas, par le test périodique qui permet de contrôler ce point.

4.4 Conclusion

La technique de test en ligne utilisant un codage par bit de parité est la technique actuellement la plus employée dans les mémoires et processeurs commerciaux et est par conséquent intéressante si on veut éviter des problèmes de compatibilité entre structures (par exemple, entre un processeur du tableau SIMD et la mémoire du processeur de contrôle). Dans notre cas, ces techniques sont assez peu coûteuses en terme de surplus matériel en raison de la prépondérance matérielle de la mémoire. Ainsi, le surplus matériel n'est que de 7,1 % pour le processeur initial de la figure 2 (tableau 7).

	déco-deur	registres	RAM	UAL	*	connexions internes	mux	contrôle parité	total
processeur 16 bits	143	3017	32768	349	504	172	52		37005
avec auto-contrôle	143	3205	34816	426	736	183	55	90*4	39781
surplus matériel	0	6,25 %	6,25 %	22 %	46 %	6,25 %	6,25 %		7,5 %
processeur 64 bits	143	11843	131072	1212	8928	652	196		154046
avec auto-contrôle	143	12028	133117	1515	15802	662	199	378*4	164978
surplus matériel	0	1,56 %	1,56 %	25 %	77 %	1,56 %	1,56 %		7,1 %

Tableau 7 : surplus matériel pour le processeur du tableau SIMD dû à l'ajout d'auto-contrôle pour le test en-ligne

Ces techniques entraînent toutefois une baisse de performance qui peut être importante (25 % de baisse en performance pour un multiplieur de Wallace 32*32 selon [Nicolaidis1997]). Ce coût peut néanmoins rester acceptable si la demande en sûreté des résultats obtenus est importante (cas de calculs de longue durée, par exemple).

La faiblesse de la structure présentée figure 7 est due à l'unique bit d'indication d'erreur. En cas de faute sur ce bit, les structures de test sont mises en échec. Il sera donc important, lors de la conception du circuit, de prévoir des règles de dessin plus larges sur ce chemin particulier. Une autre solution peut être d'utiliser des contrôleurs double-rail permettant d'obtenir un double bit de test. Cela se ferait cependant au détriment de la performance du circuit. Néanmoins, il faut noter que le bit d'indication d'erreur est lui-même périodiquement testé, ceci diminuant les risques.

5 Test des interconnexions

Le test des interconnexions entre processeurs doit être, comme pour le processeur, valable pour les phases de fabrication et de fonctionnement. La solution pour le test en-ligne a été proposée dans la section précédente, puisqu'elle supposait que tous les chemins de données des processeurs étaient munis d'un bit de parité contrôlé à chaque cycle d'horloge par le processeur.

Par ailleurs, le test des processeurs s'effectue en parallèle et de façon intégrée, c'est à dire sans utilisation des connexions extérieures. Il est donc naturel que ce test soit effectué en premier. Nous supposons qu'ensuite la reconfiguration est réalisée (voir chapitres suivants) et les interconnexions physiques entre processeurs corrects sont créées. Ce sont ces connexions obtenues à la fin de la phase de reconfiguration qu'il s'agit maintenant de vérifier.

Les fautes pouvant affecter les interconnexions sont modélisables par :

- un collage à 0 ou 1 d'une ligne unique ;
- un collage à 0 ou 1 de plusieurs lignes ;
- l'influence d'une ligne sur une autre (couplage de transition ou d'état) ;
- l'influence de plusieurs lignes sur une autre.

Les modèles de fautes d'influence des interconnexions se rapprochent de ceux des éléments de mémoire concernant les fautes d'influence entre cellules. Deux méthodes de test peuvent alors être employées :

- l'utilisation des vecteurs de test pseudo-exhaustifs pour la couverture des fautes de collage d'une et de plusieurs lignes ;
- l'utilisation des techniques de test transparent orienté mot pour la couverture des fautes d'influence entre 2 lignes. Le mot utilisé peut être choisi indifféremment dans la mémoire ou dans un registre.

Les fautes d'influences multiples ne sont pas traitées explicitement par les deux tests précédents. On peut toutefois estimer que ces deux tests assez complémentaires permettent de couvrir un grand nombre de ces fautes. Une telle affirmation reste à valider par des tests supplémentaires.

6 Synthèse générale du test d'un calculateur SIMD ou multi-SIMD

Les méthodes de test pour un processeur d'un tableau SIMD ainsi que pour les interconnexions entre processeurs ont été présentées. Ces méthodes permettent de réaliser le test après fonderie, le test périodique et le test concurrent de ces structures et assurent la couverture de fautes permanentes et transitoires.

La méthode de test de fabrication est également utilisable pour un test périodique pendant le fonctionnement du calculateur. L'approche utilisée possède de nombreux avantages. Parmi ceux-ci, on peut citer :

- la transparence des méthodes de test vis-à-vis de l'utilisateur : le test étant intégré, il n'y a pas de génération des vecteurs de test ou de vérification des résultats à faire depuis l'extérieur du système ;
- la simplicité de mise en place de la procédure de test (c'est un simple programme) ;
- la minimisation du surplus matériel et des pertes en performance ;
- la bonne couverture de fautes du point de vue fonctionnel et structurel avec un test effectué à la vitesse nominale du système et une prise en compte des fautes non modélisées ;
- la rapidité de la méthode de test pour le test périodique : premièrement, les processeurs sont testés en parallèle et deuxièmement, il n'y a pas de sauvegarde de contexte à chaque procédure de test grâce à la caractéristique de test transparent ;
- l'extension possible de la méthode de test à toutes les parties calculatoires et à tous les blocs de mémorisation ayant accès au chemin de données, et donc à des processeurs plus complexes que celui présenté figure 2 ;
- l'adaptation de cette méthode à des structures décrites sous forme de description VHDL et d'IP logicielles, ou constituées de blocs matériels optimisés (multiplieurs,...) ;

Cependant, la méthode a ses inconvénients :

- les parties séquentielles ne sont pas prises en compte dans le schéma global ;
- Pas de connaissance de la couverture réelle des défauts du système.

Le premier inconvénient mène à une première perspective possible de ce travail. Celle-ci consisterait à proposer un test périodique et concurrent des parties séquentielles relié au test d’intégrité du processeur.

Le deuxième inconvénient se pose pour tous les modèles de faute et il est très difficile d'y apporter une réponse. En effet, il faudrait disposer d'un simulateur de tous les défauts possibles pouvant affecter un circuit intégré. D'autre part, la volonté d'indépendance de la méthode vis-à-vis de l'implémentation physique des blocs implique que la réponse à la question posée ne peut être donnée qu'après la synthèse du circuit.

Afin de prendre en compte les fautes transitoires, un test concurrent de type auto-contrôle peut être associé au test périodique pour aboutir, en sortie du processeur, à un bit d'état permanent du

processeur. Ce test, qui a été choisi en raison de ses qualités de conformité avec un certain nombre de nos objectifs de test (voir la conclusion du chapitre 2), ne peut être utilisé pour un test de fabrication en raison de l'hypothèse de faute unique, mais permet de résoudre un certain nombre de problèmes liés au test des fautes transitoires. Ses caractéristiques sont les suivantes :

- surplus matériel acceptable ;
- pertes en performance limitées (en comparaison des autres méthodes) ;
- test efficace pour traiter les fautes d'interconnexion, de mémoire et de chemin de données ;
- compatibilité avec les tests en ligne de nombreuses structures (mémoires, processeurs du commerce,...)

Par contre, l'inconvénient majeur de cette méthode de test est la nécessité de modifier la structure même des blocs. Cet inconvénient peut nous amener à abandonner cette méthode lorsque le processeur est sous forme d'IP logicielle, car les modifications requises sont trop importantes, et de s'en remettre à un test concurrent sous forme logicielle, plus coûteux en performance, mais ne demandant pas de modification matérielle.

De manière plus générale, à notre avis, les principales difficultés pour les années futures viendront de :

- la définition de modèles de fautes réalistes pour les nouvelles technologies : fautes au comportement séquentiel ou transitoire bref ainsi que fautes multiples dues à des phénomènes électromagnétiques incluant plusieurs pistes et transistors.
- la définition de méthodes efficaces pour couvrir ces fautes, celles-ci pouvant présenter un caractère « exotique » (fautes d'influence, niveau logique non défini,...). Ces méthodes devront également prendre en compte la volonté de mise en place à un niveau d'abstraction important (description comportementale ou architecturale).

Les évolutions technologiques apportent donc au test des problèmes importants qu'il faudra surmonter, le test devant à la fois se rapprocher de la structure pour décrire des phénomènes physiques plus complexes, et s'en éloigner pour offrir des solutions moins coûteuses.

Chapitre 4 : état de l'art sur la reconfiguration des calculateurs parallèles

1 Introduction

Nous avons vu dans le premier chapitre différentes solutions permettant d'assurer la fiabilité des calculateurs parallèles par tolérance aux fautes. Parmi les 7 méthodes proposées, nous avons retenu la tolérance aux fautes matérielle dynamique appelée également tolérance aux fautes structurelle. La méthode générale établissant cette tolérance aux fautes a alors été donnée. Nous avons vu dans les chapitres 2 et 3 comment régler le problème du test de la structure et de la localisation des fautes par localisation du processeur fautif.

Nous nous replaçons ici dans le contexte d'un ensemble de blocs connectés entre eux par un réseau de topologie quelconque et possédant des moyens de test intégré et/ou d'auto-test permettant d'obtenir l'état de chaque bloc. Par exemple, pour le calculateur présenté au chapitre 3, les blocs sont des processeurs et le bit de test d'intégrité indique l'état de chaque processeur. A partir de ces informations, lorsque la faute est déclarée comme étant permanente, on doit opérer une reconfiguration de la structure. La reconfiguration consiste à remplacer les blocs fautifs par des blocs supplémentaires de telle façon que la topologie globale du système soit conservée. Le taux de reconfiguration est alors défini comme la probabilité qu'une structure se reconfigure correctement lorsque certains de ces processeurs ou interconnexions sont fautifs.

Les méthodes de reconfiguration ont été largement étudiées dans la littérature, mais sous un angle différent de celui sous lequel nous nous plaçons, puisqu'elles concernent soit les super-calculateurs de type MIMD, soit les Wafer Scale Integration (WSI). Le but de cette partie est donc de faire le point sur les techniques de reconfiguration pour pouvoir les comparer et établir le bilan de leurs points forts et de leurs points faibles.

Afin de se donner des moyens de comparaison, il est toujours intéressant de pouvoir classer les méthodes. Chean et Fortes proposent dans [Chean and Fortes1990] une taxonomie des solutions de reconfiguration. C'est cette taxonomie qui est maintenant reprise.

2 Taxonomie de Chean et Fortes

La taxonomie de Chean et Fortes se présente sous la forme d'un arbre binaire (figure 1). La racine de cet arbre est la redondance matérielle, laquelle redondance peut être globale ou locale. Une redondance globale autorise chaque processeur ou ensemble de processeurs à être remplacé par n'importe quel processeur ou ensemble de processeurs de la structure. Au contraire, une redondance locale limite les processeurs pouvant être utilisés pour le remplacement d'un processeur fautif ou d'un ensemble de processeurs à un environnement réduit autour des processeurs.

A partir des deux branches ainsi obtenues, la distinction est faite sur le matériel remplacé lors de la reconfiguration. Ce matériel peut être un processeur unique fautif (reconfiguration à grain fin), ou un ensemble de processeurs comprenant le processeur fautif (reconfiguration à gros grain).

Seules 3 des 4 branches auxquelles on peut ainsi aboutir sont considérées par la suite, la quatrième branche ne présentant aucun intérêt. Pour chacune de celles-ci, le remplacement matériel peut être réalisé de façon globale (un maître programme les aiguillages) ou locale (chaque aiguillage se programme indépendamment des autres).

Enfin, pour chacune des nouvelles branches, il existe différentes possibilités d'implémentation de la commutation. Celle-ci peut être réalisée soit par un élément d'aiguillage, soit par bus, soit par un réseau d'aiguillage. Dans le premier cas, l'aiguillage est formé d'éléments comme des multiplexeurs à l'intérieur des processeurs. Les méthodes de by-pass d'un processeur utilisent de tels systèmes. Dans le deuxième cas, on réalise une commutation de circuit, les bus pouvant être connectés entre eux de différentes manières. C'est le cas le plus courant. Un aiguillage par réseau s'apparente plus aux techniques utilisées pour rendre un réseau fonctionnellement reconfigurable, par l'utilisation, par exemple, de crossbar ou de réseaux programmables dynamiquement comme les réseaux FFT.

La figure 1 reprend la taxonomie de Chean-Fortes en associant aux branches correspondantes les schémas de reconfiguration principaux étudiés par la suite.

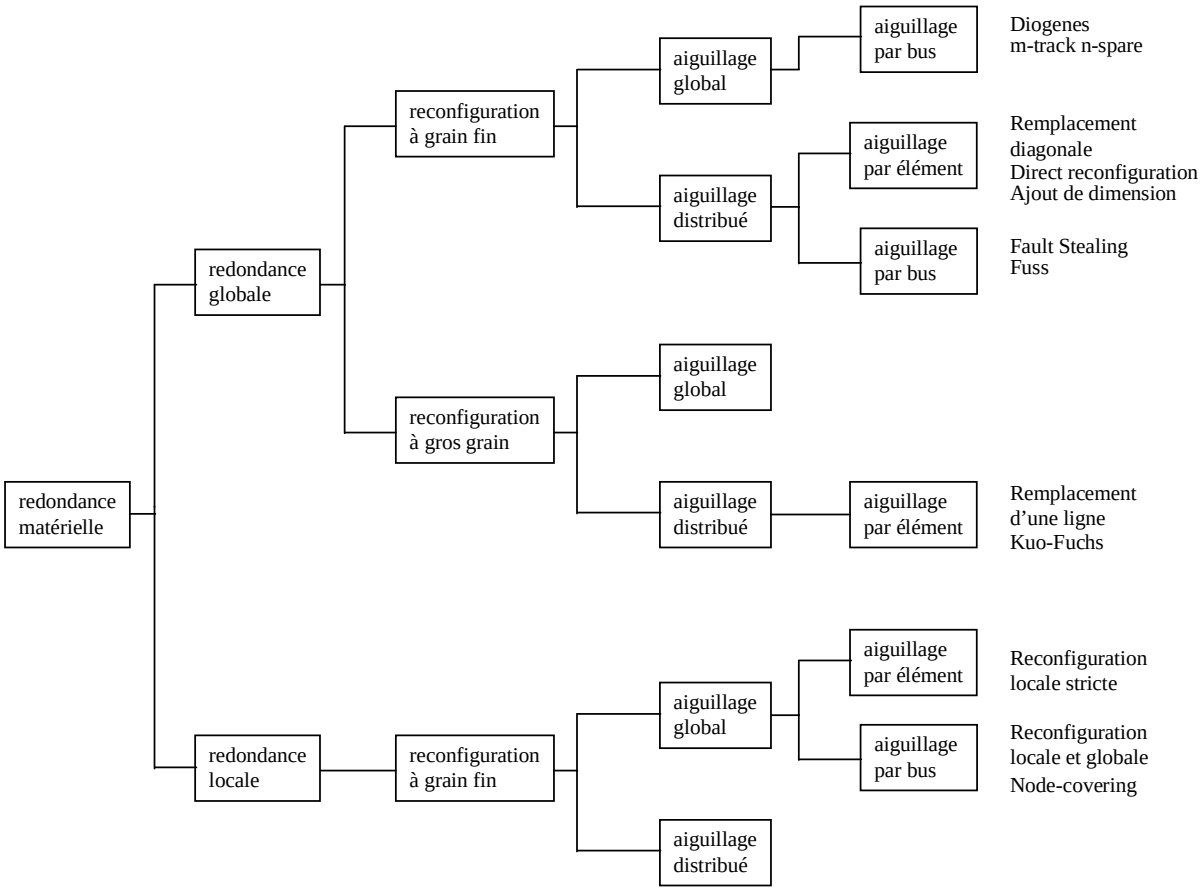


Figure 1 : taxonomie de Chean-Fortes et schémas de reconfiguration correspondant aux feuilles de l'arbre

Cette taxonomie est utile pour classer les différents schémas de reconfiguration et donner une vue générale des grandes lignes de la reconfiguration. Malheureusement, elle ne permet pas de prendre en compte les facteurs influant sur le choix d'une solution plutôt qu'une autre. Ainsi, des méthodes très différentes et ayant des avantages et inconvénients pratiquement opposés se retrouvent dans la même branche de la taxonomie, alors que d'autres branches sont vides, car le schéma qu'on pourrait en déduire ne présente aucun intérêt.

Dans la suite de cette présentation, et afin de conserver la clarté nécessaire pour pouvoir comparer les méthodes, nous nous limiterons donc à distinguer deux branches de l'arbre qui représentent chacune un ensemble d'avantages et d'inconvénients communs. Ces deux branches concernent le matériel qui va être remplacé par la reconfiguration, et donc les schémas de reconfiguration à gros grain et à grain fin. De plus, dans chaque catégorie, nous nous limiterons aux schémas qui nous paraissent les plus intéressants (les plus proches de notre approche) parmi la multitude de techniques proposées.

3 Reconfiguration à gros grain

La reconfiguration à gros grain consiste à prendre un ensemble comprenant l'élément fautif afin de le remplacer par un autre ensemble d'éléments corrects. C'est la forme la plus simple de reconfiguration et ce sont les premiers schémas à avoir été étudiés. Ceux-ci ont été beaucoup utilisés pour les intégrations sur tranche (WSI). La reconfiguration à gros grain possède les avantages de simplicité des méthodes, de rapidité de la reconfiguration ainsi que de limitation de la longueur des interconnexions entre éléments voisins. Par contre, les taux de reconfiguration sont faibles à très faibles, car un ensemble d'éléments supplémentaires est « gaspillé » pour couvrir une seule faute.

La solution la plus simple de ce type de reconfiguration est le remplacement d'une ligne (ou d'une colonne) de processeurs, placée généralement à la frontière du réseau [Batcher1980, Koren1981]. Cette solution, très simple dans la mise en oeuvre est utilisée dans quelques réseaux tels que PAPIA2, le « polymorphic-torus network » ou le « gated-connection network » [Li and Stout1991].

Pour des approches WSI et lorsque le nombre d'éléments est important, il est possible d'ajouter plusieurs lignes et/ou colonnes d'éléments supplémentaires. Il faut alors optimiser l'utilisation de ces lignes et colonnes afin d'assurer la meilleure reconfiguration possible. Il a été prouvé que le problème était NP-complet par Kuo et Fuchs [Kuo and Fuchs1987] qui proposent une stratégie d'approximation de l'algorithme idéal pour les gros réseaux.

De nombreuses autres approches ont été développées pour les WSI [Leighton and Leiserson1986, Moore and Mahat1985]. Toutefois, nous nous limiterons à cette brève introduction, car d'une part, ces méthodes sont très nombreuses et il serait trop long et fastidieux de les décrire toutes ici. D'autre part, le schéma de reconfiguration à gros grain ne sera retenu que dans sa forme la plus simple comme nous le verrons dans le chapitre 6, car il entraîne un ratio surplus matériel/taux de reconfiguration qui n'est pas acceptable.

4 Reconfiguration à grain fin

4.1 Introduction

Nous considérons, dans le cas des calculateurs parallèles, que l'élément de base est le processeur. Un schéma de reconfiguration basé sur le remplacement d'un processeur fautif par un processeur supplémentaire, de façon directe ou indirecte, sera donc dit à grain fin.

L'avantage principal de ces méthodes vient du fait que les processeurs supplémentaires sont, a priori, moins « gaspillés » que pour un schéma à gros grain. Cette approche sera donc préférée à la précédente lorsqu'on attache plus d'importance à l'élément de base. La conséquence est que les réseaux et/ou les méthodes sont, en général, plus complexes.

La plus simple de ces approches est dite de « remplacement diagonal » [Li and Stout1991]. La méthode est la suivante : les processeurs de la diagonale du réseau sont utilisés comme processeurs supplémentaires. Des éléments de connexion appelés « switches » et des interconnexions sont alors ajoutés de façon à pouvoir remplacer de manière logique chaque processeur du réseau par le processeur de la diagonale qui lui correspond. Ce schéma permet un meilleur rendement d'utilisation des processeurs supplémentaires que le schéma à gros grain. Par contre, ce dernier devient difficile à mettre en oeuvre pour des systèmes comportant un grand nombre de processeurs appartenant à plusieurs cartes. Par ailleurs, il ne donne pas entière satisfaction au niveau du rapport entre le matériel ajouté et l'amélioration de la fiabilité obtenue.

Dans la suite, nous nous focalisons sur des méthodes apportant une bonne utilisation des processeurs supplémentaires. Parmi ces méthodes, nous distinguons l'ajout de dimension, l'approche Diogenes, les méthodes par remplacements successifs et les schémas de reconfiguration locale où un processeur ne peut être remplacé que par un sous-ensemble des processeurs supplémentaires.

4.2 Reconfiguration par ajout de dimension

L'utilisation d'un réseau de dimension plus important que celui que l'on désire est une solution de reconfiguration qui peut être envisagée [Bae and Bose1996, Bruck1993, Bruck1992, Raghavendra1992, Raghavendra1995]. On peut considérer, par exemple, la grille 2-D comme un sous-réseau d'hypercube. La reconfiguration consiste alors à plonger le réseau logique que l'on souhaite obtenir (la grille) dans le réseau physique (l'hypercube) comportant les processeurs fautifs.

Cette solution est intéressante, mais présente des inconvénients majeurs. Le premier est une complexité accrue du schéma de reconfiguration, car on doit trouver, à chaque faute détectée, un plongement (au sens de la théorie des graphes) de la structure logique sur la structure physique. Ensuite, le nombre de processeurs supplémentaires ne peut pas être choisi, ni finement, ni grossièrement (comme l'ajout d'une ligne de processeurs en plus) dans certains de ces schémas. Mais l'inconvénient majeur est que cette méthode ne prend pas en compte les possibilités de réalisation physique. Ainsi, elle implique une augmentation de la complexité de la structure, notamment un accroissement du nombre des ports des processeurs. Par ailleurs, des interconnexions de longueur importante doivent être ajoutées et la proximité physique de processeurs voisins logiques

n'est pas respectée. Ainsi, une grille 2-D perd certaines de ses propriétés intéressantes au niveau réalisation physique, comme le nombre de ports limité à 4 ou sa modularité. Cette solution ne peut donc pas être adoptée dans notre cas.

4.3 L'approche Diogenes

L'approche dite de Diogenes [Rosenberg1983] est une approche originale de reconfiguration, qui vise à obtenir 100% de reconfiguration tout en gardant une grande simplicité de programmation de la configuration. En effet, celle-ci est obtenue de façon matérielle à partir de quelques informations essentielles comme l'état du processeur et sa position topologique pour certains réseaux (frontière, feuille ou racine d'un arbre, ...).

La première phase d'une approche Diogenes consiste à rendre linéaire le tableau de processeur au niveau du schéma physique, qu'il s'agisse d'un arbre, d'une pyramide, d'un maillage ou tout simplement d'un anneau de processeurs. A partir de la structure linéaire, un « paquet » (bundle) de fils composé d'un certain nombre de bus (déterminé par la topologie et le nombre de processeurs) est mis en place. Un système d'aiguillage basé sur des principes de piles logiques et/ou de queues logiques permet ensuite de déterminer les connexions des processeurs sur les bus.

Afin d'expliciter ce schéma, nous allons prendre l'exemple très simple de l'anneau. C'est une structure linéaire qui ne demande donc pas de linéarisation. La figure 2 montre la structure ainsi que la cellule élémentaire, très simple dans ce cas, permettant la reconfiguration. Cette cellule est composée d'un système de mémorisation de la programmation (latch) et d'un système rudimentaire d'aiguillage composé de transistors bloquant/passant. La seule information nécessaire est l'état du processeur.

L'approche Diogenes peut ainsi être adaptée à diverses topologies 2-D (grille, arbre, pyramide) avec une complexité plus importante, mais toujours assez faible. Ce système de reconfiguration permet d'obtenir 100 % de taux de reconfiguration pour un nombre quelconque de processeurs supplémentaires. Enfin, le nombre d'entrées/sorties des processeurs reste inchangé. Tous ces avantages en font un bon candidat à la reconfiguration.

Toutefois, cette approche présente des inconvénients liés à la linéarisation nécessaire des processeurs. La figure 3 montre un exemple de linéarisation d'un maillage composé de 16 processeurs seulement et les liens entre ces processeurs en l'absence de faute. Le premier problème, comme le signal l'auteur, concerne la forme physique imposée des processeurs [Rosenberg1983]. Ces processeurs doivent ainsi être allongés et très étroits pour une implémentation sur ASIC, ce qui est difficilement acceptable dans une phase de placement/routage. Le deuxième inconvénient majeur concerne la longueur des interconnexions. On peut en effet constater, sur la figure 3, que la longueur des interconnexions entre deux processeurs voisins logiquement est déjà importante sans processeur fautif. Cette distance est ensuite susceptible d'augmenter en fonction des fautes du circuit jusqu'à devenir très importante, posant des problèmes au concepteur qui doit alors choisir soit de dimensionner la fréquence des communications en fonction du chemin le plus long, entraînant ainsi de fortes pertes en performance, soit de limiter la longueur possible, faisant alors chuter le taux de

reconfiguration. Enfin, les interconnexions sont supposées non fautives et il paraît difficile d'ajouter à la méthode une façon de tolérer ces fautes.

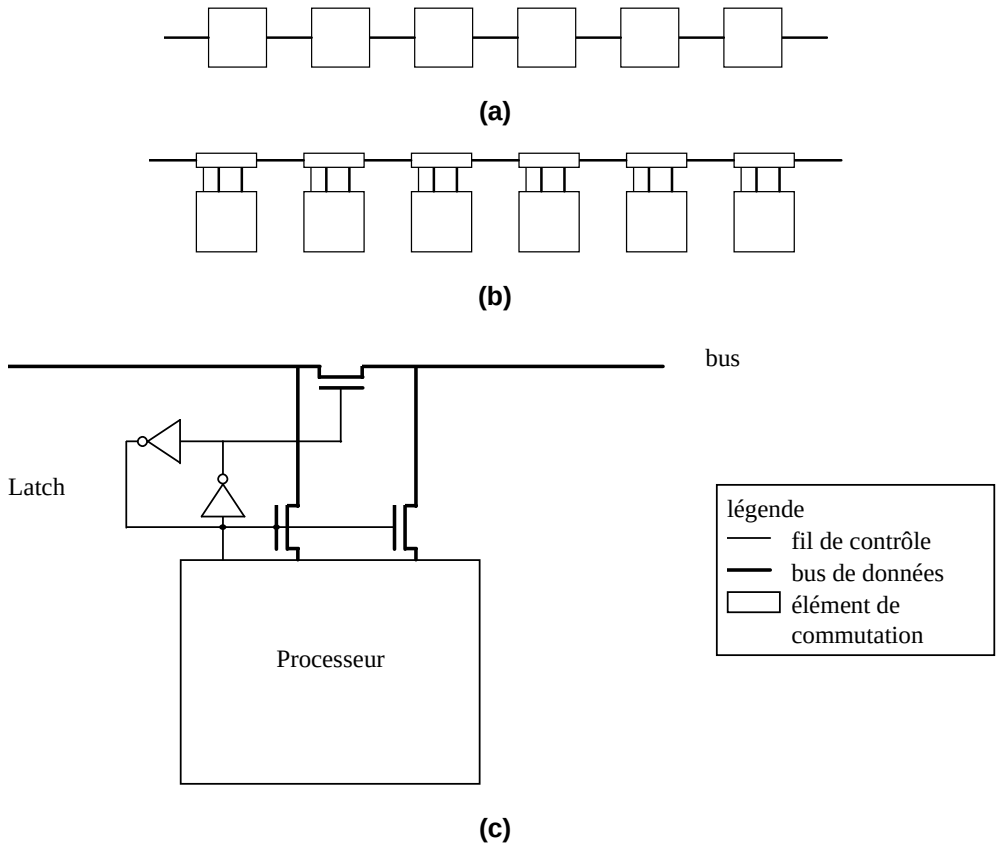


Figure 2 : exemple d'une approche Diogenes. (a) le tableau de processeur une dimension. (b) son implémentation physique par approche Diogenes. (c) détail d'une cellule de reconfiguration

Deux schémas pour les topologies de maillage ont été proposés plus récemment dans [Belkhale and Banerjee1992] afin d'amoindrir les défauts cités ci-dessus. Le premier schéma conserve l'implémentation physique du maillage en deux dimensions et considère une approche Diogenes sur chaque ligne et chaque colonne, permettant de réduire considérablement les longueurs d'interconnexion et la surface supplémentaire. Mais la conséquence est une chute du taux de reconfiguration qui devient faible.

Le second schéma propose une linéarisation du maillage moins importante que celle de l'approche normale. Pour 16 processeurs, par exemple, le réseau est implémenté sur 2 lignes de 8 processeurs. Cette approche conserve des taux de reconfiguration assez bons, tout en diminuant également fortement le surplus matériel. Par contre, le problème de la longueur des interconnexions entre processeurs voisins persiste alors que la réalisation physique de ce réseau n'apparaît toujours pas aisée. Enfin, la modularité de l'approche est perdue.

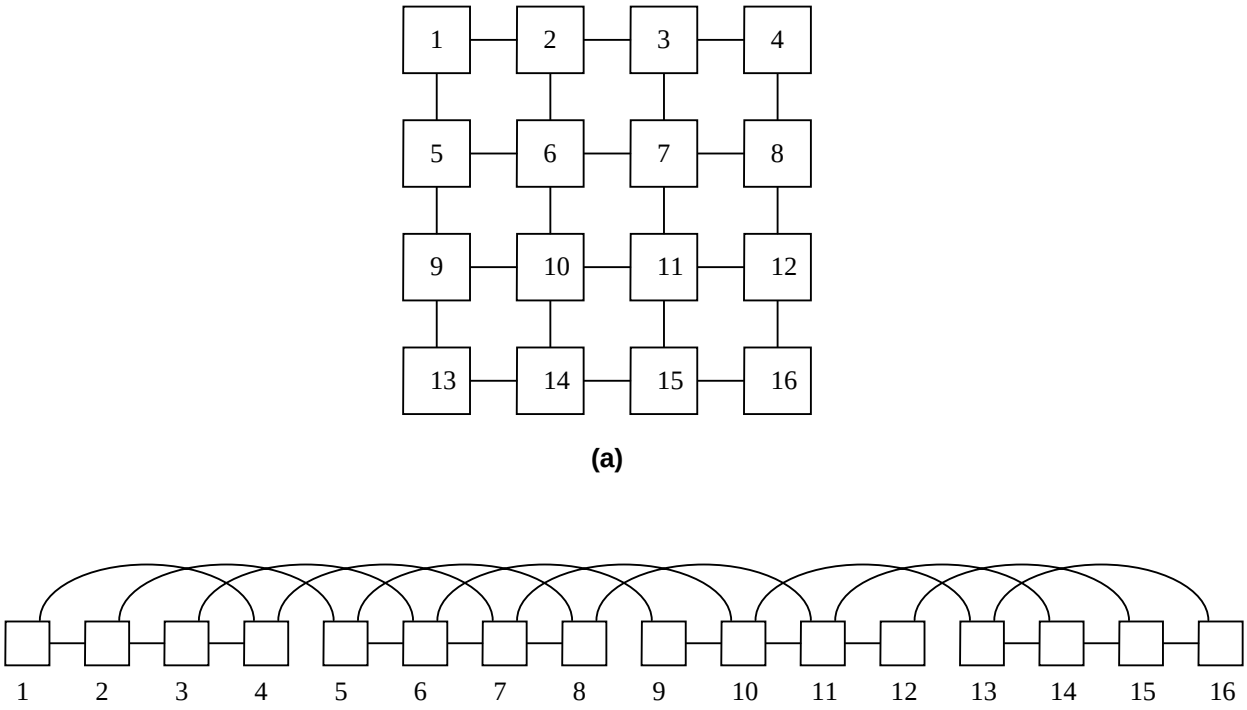


Figure 3 : (a) réseau maillage 2-D 4*4 processeurs et (b) son implémentation physique par approche Diogenes

Pour conclure sur l'approche Diogenes, on peut dire que cette méthode permet d'optimiser théoriquement la durée de l'algorithme et les probabilités de reconfiguration, mais qu'une implémentation paraît difficile à cause des longueurs des interconnexions et de la forme physique (layout) nécessaire des processeurs pour une intégration VLSI. Ce problème est en général critique dans les calculateurs, et encore plus particulièrement pour les calculateurs de type SIMD où les communications régulières constituent le talon d'Achille.

4.4 Les approches par remplacements successifs

4.4.1 Introduction

Les méthodes de reconfigurations par remplacements successifs ont fait l'objet de nombreuses études [Jean1990, Jean and Kung1989, Kung1989, Lombardi1989, Mahapatra and Dutt1996, Roychowdhury1990, Varvarigou1991, Varvarigou1993]. Ces méthodes utilisent toutes le même principe qui est de remplacer un processeur fautif par un de ses voisins, puis le voisin par un de ses voisins, et ainsi de suite jusqu'à un processeur supplémentaire.

Parmi ces méthodes, nous nous intéressons aux premières méthodes développées de « fault-stealing », « direct-reconfiguration » et algorithme FUSS, ainsi qu'aux réseaux m-track n-spare, dont les plus récents développements montrent d'excellentes qualités de reconfiguration.

4.4.2 Les méthodes de « fault-stealing » et « direct reconfiguration » et algorithme FUSS

La méthode dite de reconfiguration directe [Negrini1986] est la plus simple des méthodes de remplacements successifs. Elle utilise une colonne et une ligne de processeurs supplémentaires. L'algorithme consiste à balayer chaque colonne de bas en haut en marquant le premier élément fautif (s'il existe) comme une faute verticale, et les autres éléments fautifs de la colonne comme des fautes horizontales. Les fautes horizontales sont alors remplacées par l'élément de droite alors que les fautes verticales sont remplacées par l'élément en bas. On recommence alors le balayage et la phase de décalage jusqu'à atteindre les processeurs supplémentaires (figure 4). L'algorithme échoue dès lors qu'il y a plus d'une faute horizontale.

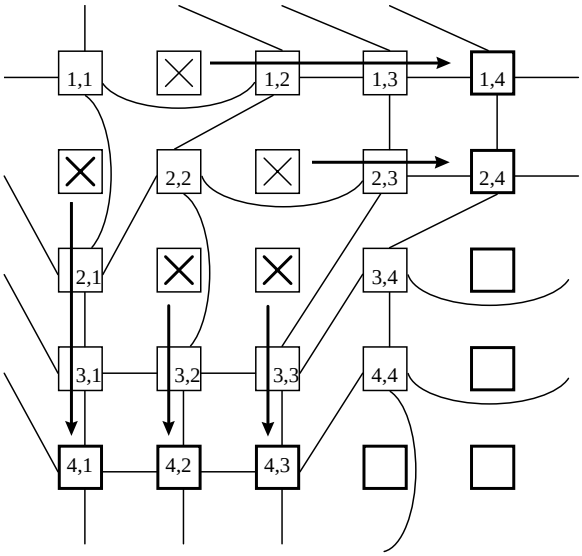


Figure 4 : exemple de technique par couverture successive, la reconfiguration directe

Ce schéma simple ne donne pas de très bons résultats de reconfiguration et peut être amélioré en donnant une plus grande liberté au choix des processeurs pouvant remplacer un processeur fautif. La généralisation des méthodes de « direct reconfiguration » est ainsi appelée « fault-stealing » [Negrini1986]. Ces schémas plus complexes induisent également des architectures plus complexes comportant un nombre d'interconnexions croissant.

Une autre stratégie est l'algorithme FUSS (Full Use of Suitable Spares) [Chean and Fortes1989], qui permet une reconfiguration presque optimale. Le réseau comporte des colonnes supplémentaires, et l'algorithme se déroule en deux étapes : la première consiste à répartir certains processeurs fautifs des lignes dont le nombre de processeurs fautifs empêchent la reconfiguration sur les lignes adjacentes. Une technique de direct reconfiguration est alors appliquée.

Ces méthodes sont relativement simples à mettre en œuvre et présentent de bons résultats de reconfiguration. Les inconvénients au niveau de la longueur des interconnexions entre processeurs voisins subsistent toutefois. D'autre part, le nombre de processeurs supplémentaires doit être au minimum d'une ligne et d'une colonne pour le fault-stealing et d'une colonne pour l'algorithme FUSS.

Enfin, ces schémas de reconfiguration imposent une architecture et ne tiennent pas compte des fautes pouvant survenir dans les interconnexions.

4.4.3 Les réseaux n-track m-spare

Le « schéma multi-chemins » (« multi-tracks scheme ») [Jean1990, Jean and Kung1989, Kung1989, Mahapatra and Dutt1996, Roychowdhury1990, Varvarigou1991, Varvarigou1993] utilise une autre solution de couverture successive des processeurs. La structure générale est formée par un réseau comportant une à plusieurs lignes (ou colonnes) de processeurs, dits processeurs « spare » aux frontières du réseau. Des moyens d'interconnexion entre les lignes et les colonnes de processeurs, appelés « tracks » et reliés au réseau par des switches sont alors ajoutés (voir figure 5). On parlera ainsi de 2-track 1-spare pour un réseau auquel on a ajouté deux chemins d'interconnexion entre les lignes et entre les colonnes de processeurs et une ligne/colonne de processeurs spare à chaque frontière du réseau. On peut généraliser en parlant de réseau m-track n-spare lorsque qu'on a n lignes/colonnes de processeurs spare à chaque frontière du réseau et m ligne/colonne d'interconnexions entre les lignes/colonnes de processeurs du réseau. De plus, si pour un modèle m-track n-spare, on ajoute la possibilité de faire le court-circuit des processeurs, on parle alors de m½-track, n-spare.

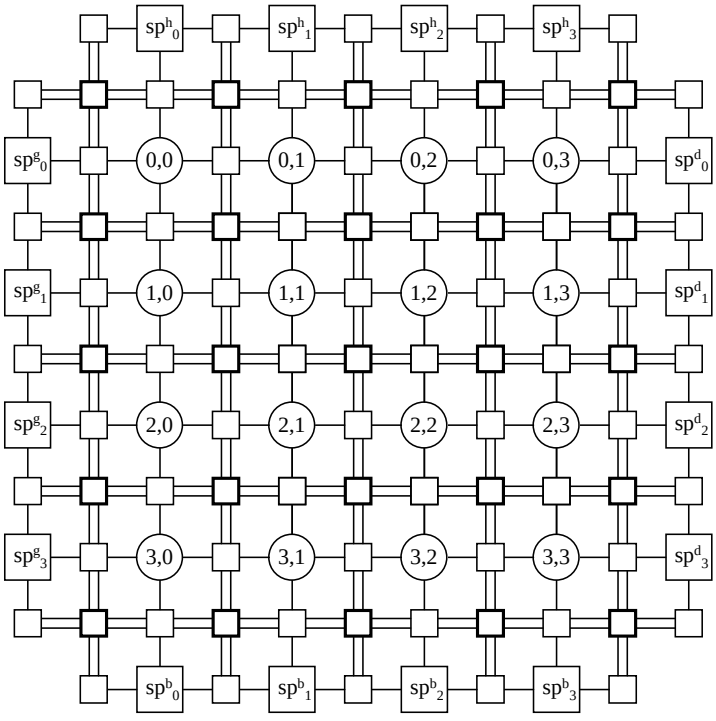


Figure 5 : réseau reconfigurable du type « schéma multi-chemins ». Les processeurs supplémentaires (sp) entourent le réseau initial, et deux types d'éléments d'interconnexions appelés switches sont utilisés, ce sont les PT (Processor to Track) indiqués en trait fin et les TT (Track to Track) en traits gras

La reconfiguration s'effectue en deux étapes. Lorsqu'un processeur fautif est détecté, on cherche à remplacer ce processeur par un de ses voisins, de façon à conserver la continuité dans le réseau physique. Ce processeur est donc le remplaçant « logique » du processeur fautif. Il doit à son tour être remplacé dans le réseau par un de ses voisins, et ainsi de suite jusqu'à ce que le processeur remplaçant soit un processeur « spare ». Cette recherche du remplacement de chaque processeur constitue la première étape et s'appelle la recherche du chemin de compensation du processeur fautif (« compensation path »). La deuxième étape consiste alors à effectuer le routage des switches du réseau de façon à connecter les processeurs placés logiquement sur le réseau dans l'étape précédente.

Les capacités matérielles d'interconnexions limitent les possibilités de reconfiguration de ces réseaux. Selon leur nombre, la méthode de compensation employée et les éléments d'interconnexion utilisés, le placement des processeurs pourra ou non se faire en respectant certaines règles qui impliquent la réussite du routage. Bien entendu, plus on a de liens ou « tracks », plus les chances de routage augmentent, mais aussi plus le système est lourd et difficile à construire et l'algorithme de placement compliqué. Un équilibre est alors à trouver. Les solutions de ce type peuvent être utilisées dans un grand nombre d'architectures statiques comprenant les hypercubes, les grilles ou autres structures régulières.

4.4.4 Conclusion

Nous avons vu que les méthodes de remplacements successifs ont été améliorées jusqu'à atteindre un niveau de sophistication important, permettant une excellente reconfiguration pour un coût matériel correct. De plus, les dernières implémentations de ces méthodes sont adaptables à différents types de réseaux, elles sont bien maîtrisées et les éléments théoriques les composant sont connus.

Toutefois, les mêmes défauts récurrents, dus à l'approche suivie, apparaissent dans tous ces schémas :

- la longueur des interconnexions entre processeurs voisins ;
- la non-modularité du schéma de tolérance aux fautes ;
- le fait d'imposer une architecture pour la reconfiguration ;
- l'absence de prise en compte de possibles fautes au niveau des interconnexions.

4.5 Les réseaux reconfigurables au niveau local

4.5.1 Introduction

Pour les schémas de reconfiguration locale, chaque processeur fautif ne peut être remplacé que par un nombre limité d'éléments supplémentaires. Les avantages généraux de ces méthodes sont multiples, puisqu'elles permettent de conserver une meilleure localité, et donc de limiter la longueur des interconnexions entre processeurs voisins, elles sont en général peu coûteuses en bande passante et en matériel supplémentaire. Ceci est obtenu en conservant un taux de reconfiguration correct mais largement inférieur aux techniques de reconfiguration globale.

Leur principal défaut est dû à la localité qui entraîne une perte de fiabilité lorsqu'elle est appliquée au sens strict du terme. Par conséquent, certains aménagements ont été faits afin de prendre en compte ce défaut, aux dépens du surplus matériel, de la simplicité du schéma, de la bande passante nécessaire et de la longueur des interconnexions. On verra ainsi successivement une approche « simple » et stricte de la localité, suivie de deux approches, l'une à reconfiguration locale et globale, et l'autre appelée le « node-covering ».

4.5.2 Reconfiguration locale stricte

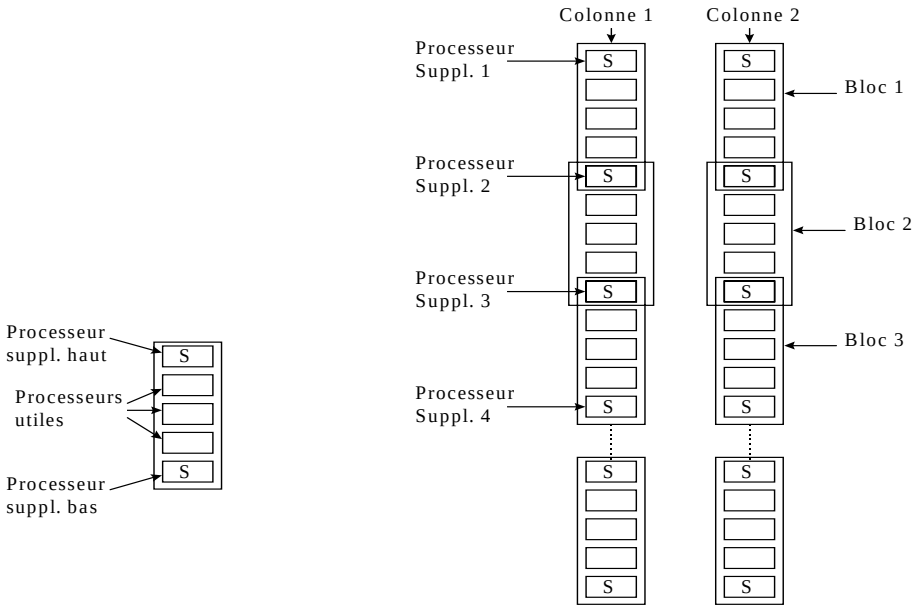
Le terme de reconfiguration locale stricte vient du fait que dans ce schéma, un processeur supplémentaire remplace directement le processeur fautif [Singh1988]. Pour ce faire, un certain nombre de processeurs du réseau sont doublés d'un processeur supplémentaire ou « spare ». Les processeurs voisins peuvent alors être remplacés à moindre coût par ce doublon en cas de faute. Le problème revient alors à placer correctement les processeurs supplémentaires dans le réseau.

Cette solution, très peu gourmande en matériel et relativement simple à mettre en oeuvre, possède toutefois l'inconvénient de ne pas assurer la reconfiguration du réseau en cas de groupement de fautes en un point du circuit (cas de surchauffe locale, par exemple). Cette solution est également peu modulable au niveau du choix de la fiabilité désirée et du choix des interconnexions à ajouter. De plus, l'implémentation physique de ce type de réseau n'apparaît pas évidente, tant au niveau du placement des processeurs supplémentaires que de la gestion des interconnexions supplémentaires, à moins de conserver un niveau complet du circuit dédié à la tolérance aux fautes.

4.5.3 Reconfiguration locale et globale

La reconfiguration locale stricte limite donc l'amélioration de la fiabilité du réseau. Pour parer à ce handicap, une solution proposée par [Chen1997] est d'utiliser une reconfiguration locale avec un certain nombre de processeurs supplémentaires permettant de remplacer chaque processeur. Le réseau en tableau 2-D d'origine est divisé en blocs, chacun comprenant une ligne (ou une colonne) de processeurs ayant un processeur supplémentaire à chaque extrémité. Le réseau global est alors obtenu par jonction des différents blocs, comme le montre la figure 6.

Deux algorithmes de remplacement des processeurs fautifs par les processeurs supplémentaires permettent de déduire deux réseaux d'interrupteurs (switches) pour la reconfiguration, comportant un nombre d'interconnexions supplémentaires assez important (6 pour le premier, 8 pour le second). Le premier algorithme prévoit la possibilité de remplacer chaque processeur utile par les processeurs supplémentaires supérieurs et inférieurs des blocs auxquels il appartient. Pour assurer la reconfiguration, un réseau de taille importante est déduit. Le deuxième algorithme ajoute aux possibilités offertes par le premier algorithme des possibilités de remplacement avec un bloc à la droite du bloc auquel appartient le processeur. Le réseau déduit est alors encore plus important que le premier.



a) bloc tolérant aux fautes **b)** structure résultant de l'association des blocs

Figure 6 : structure pour la reconfiguration locale et globale.

L'étape suivante consiste à placer les processeurs logiques sur le réseau physique, puis une table dite « de guidage » est réalisée. Celle-ci relie les processeurs selon les possibilités d'interconnexion et en tenant compte d'éventuelles fautes au niveau des switches. Cette méthode permet, d'une part, de router les liens d'interconnexion, et d'autre part à faire une analyse de la dégradation des performances, en regardant la longueur des chemins de reconfiguration. Grâce à la reconfiguration locale, cette longueur reste en général assez faible.

L'inconvénient majeur de cette méthode est le nombre très important d'interconnexions nécessaires. Par ailleurs, un nombre minimal de processeurs est requis pour pouvoir utiliser la méthode.

4.5.4 « Node-covering »

Le « node-covering » est proposé par [Dutt and Mahapatra1997]. Pour cette approche, des codes linéaires correcteurs d'erreur (ECC : Error Correcting Codes) sont utilisés pour déterminer le partage du réseau en groupes tolérants aux fautes de façon déterministe (c'est à dire « à coup sûr »).

Un des schémas utilise le code de parité 2-D (figure 7). Pour ce faire, chaque ligne et chaque colonne de l'implémentation physique du réseau sont considérées comme un réseau local de reconfiguration. Les blocs locaux sont donc entrecroisés. Le surplus matériel est alors moins important pour un taux de reconfiguration très bon, mais le principe de localité des processeurs n'est pas conservé et le nombre de processeurs supplémentaires n'est pas modulable. Il est à noter toutefois que cette méthode permet de reconfigurer n'importe quel réseau, ce qui la rend intéressante.

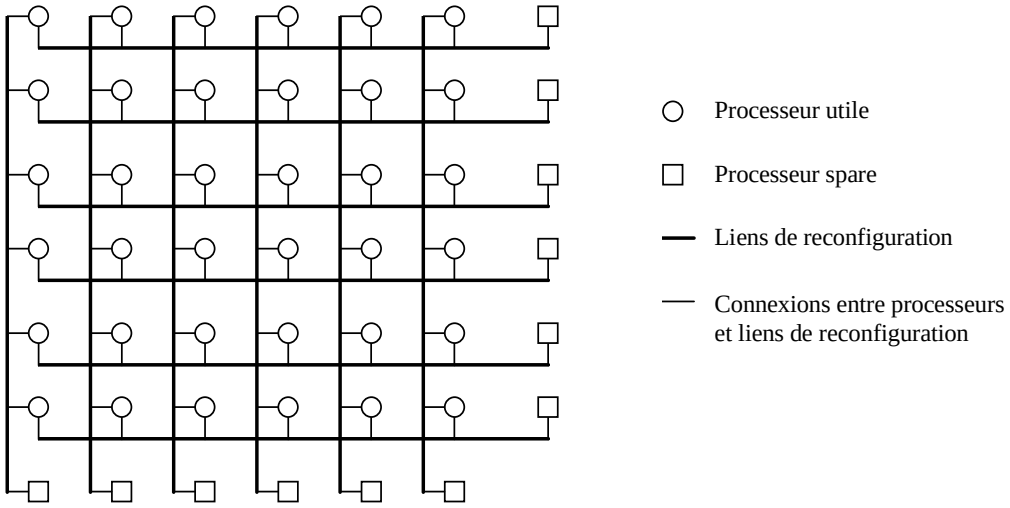


Figure 7 : architecture de node-covering par code de parité 2-D

4.5.5 Conclusion

Les méthodes de reconfiguration locales offrent des taux de reconfiguration moins importants que les méthodes globales comme l'approche Diogenes, puisque certaines configurations de fautes bien précises peuvent mettre le système en faute très rapidement. Toutefois, la probabilité d'apparition de ces configurations étant très faible, ces schémas atteignent tout de même une fiabilité moyenne très bonne pour un coût matériel moins important que pour la reconfiguration globale. Le compromis matériel-performance / amélioration de fiabilité semble donc plus facile à réaliser avec des méthodes de reconfiguration locales.

Il faut enfin parler, au niveau de la reconfiguration locale, des schémas utilisant une hiérarchie de reconfiguration. Le niveau le plus bas peut alors utiliser, par exemple, une reconfiguration à grain fin, alors que le niveau le plus haut peut utiliser, par exemple une reconfiguration à gros grain. Ces solutions sont souvent intéressantes lorsque le système n'est pas homogène dans sa conception, c'est à dire lorsque le système lui-même possède plusieurs niveaux de hiérarchie physique. C'est le cas lorsqu'une structure est réalisée à partir d'ASIC intégrant quelques dizaines de processeurs. Un double niveau de reconfiguration est donc adapté aux structures physiques envisagées pour nos futurs calculateurs.

4.6 Conclusion

Les différentes méthodes de reconfiguration à grain fin existantes ont été brièvement présentées. Ces méthodes visent toutes à une bonne utilisation des processeurs supplémentaires, mais utilisent des approches ne présentant pas les mêmes avantages et inconvénients. Ceux-ci sont discutés dans la section suivante.

5 Comparaison des différentes méthodes

Tous les réseaux reconfigurables présentés dans les sections précédentes ont été conçus dans des buts différents, de la minimisation du surplus matériel à la maximisation des possibilités de reconfiguration. Il en résulte de grandes disparités qui confèrent à chacun un certain nombre d'avantages et d'inconvénients. Le tableau 1 récapitule ceux-ci. Les critères de comparaison retenus ici tentent de rendre compte des imperfections et des réussites de chaque modèle.

Le premier critère est celui du taux de reconfiguration des différents réseaux. Celui-ci prendra en compte non seulement la reconfiguration des processeurs, mais également celle des interrupteurs (switches) ou interconnexions fautives. Ainsi, les résultats de reconfiguration seront présentés selon trois volets :

- le taux de reconfiguration moyen des processeurs, soit la probabilité de reconfiguration du réseau en fonction du nombre de fautes des processeurs (1a) ;
- les possibilités de reconfiguration lorsque des interrupteurs et/ou des interconnexions sont fautifs (1b) ;
- la « Deterministic Fault Tolerance » (DFT), soit le nombre minimal de processeurs fautifs que le réseau est sûr de reconfigurer (1c).

Le deuxième critère est le surplus matériel engendré par les différentes solutions. Ce surplus est constitué des processeurs supplémentaires ainsi que des interconnexions et des interrupteurs ajoutés au réseau originel. Nous distinguons :

- le surplus matériel en interrupteurss, processeurs et interconnexions en terme de surface (2a) ;
- l'augmentation de la bande passante, ce qui correspond à la quantité d'interconnexions supplémentaires. Une trop forte demande en bande passante supplémentaire peut rendre difficile, voire impossible, la réalisation de l'architecture reconfigurable, et/ou entamer fortement les performances des calculateurs (2b) ;
- la modularité du nombre de processeurs supplémentaires. Une bonne modularité mène à des schémas de reconfiguration permettant de mieux adapter le compromis surplus matériel / augmentation de fiabilité (2c) ;
- la facilité à transposer le schéma théorique en une réalisation physique (layout). Ce critère nous paraît très important alors que la plupart des méthodes ne le prennent pas en compte comme donnée du problème.

Le troisième critère concerne les dégradations de la performance du système. En effet l'ajout d'interrupteurs dans les chemins et l'allongement des connexions entre PE coûtent en performance. Les données sur cette dégradation ne sont pas toujours disponibles, car elle n'est pas prise en compte dans beaucoup de cas (jugée comme « faible » ou tout simplement ignorée). On peut toutefois donner des critères qualitatifs obtenus par l'étude du comportement des algorithmes sur certaines configurations bien précises de fautes (3).

Le quatrième critère concerne le type de réseau sur lequel peut s'appuyer la reconfiguration. En effet, certains schémas ne portent que sur un type précis de réseau d'interconnexions (maillage par exemple) alors que d'autres peuvent être adaptés à différents types de réseaux (4).

Enfin, le dernier critère est la facilité de reconfiguration, liée à la complexité de l'algorithme mis en oeuvre. Ce dernier critère est le moins important lorsqu'on considère une reconfiguration fine (même si le temps ne doit, bien sûr, pas dépasser une certaine limite, un algorithme mettant une heure à s'exécuter ne serait pas accepté). Il peut par contre prendre beaucoup d'importance pour les reconfigurations à gros grain qui sont utilisées dans les WSI, où la reconfiguration doit se faire « à la volée », sur la chaîne de fabrication.

Ces critères sont maintenant comparés pour les différentes approches. Au niveau des schémas de reconfiguration à gros grain, nous comparons :

- le remplacement d'une ligne ;
- l'algorithme de Kuo-Fuchs.

En ce qui concerne les schémas de reconfiguration à grain fin, seront comparées :

- l'approche par ajout de dimension ;
- l'approche Diogenes ;
- les méthodes par remplacements successifs par ces deux représentants les plus aboutis, le schéma 4-track 2-spare (1) et l'autre au surplus matériel minimisé, le schéma 2-track-1 spare (2) ;
- les méthodes de reconfiguration locale par le biais des 3 représentants que sont la reconfiguration locale stricte (1), la reconfiguration locale « améliorée » de (2) et le node-covering (3).

6 Limite des schémas de reconfiguration actuels

Il ne faut pas perdre de vue que le but de la reconfiguration est d'augmenter la fiabilité du système, donc la probabilité que ce dernier fonctionne à un temps t (ou pendant un temps t) : la reconfiguration doit permettre d'atteindre le but fixé, mais n'est pas un but en elle-même. Dans ces conditions, atteindre un taux de reconfiguration exceptionnel permettant d'aller largement au-delà de la demande en fiabilité, au détriment de la fréquence de fonctionnement, n'est pas accepté par les concepteurs d'architectures parallèles. Il est donc souvent préférable de limiter cette reconfiguration du moment qu'elle est suffisante pour la demande en amélioration de la fiabilité. Aussi, donner un taux de reconfiguration devrait toujours aller de pair avec une étude de l'amélioration de fiabilité déduite [1].

[1] Cette étude, dans notre cas, est effectuée à la fin du chapitre 6

critères / méthode de reconfiguration	tolérance aux fautes			surplus matériel				pertes en performances	réseau	Algo
	1.a	1.b	1.c	2.a	2.b	2.c	2.d	3	4	5
remplacement d'une ligne	--	nul	--	++	-	++	++	++	maillage	++
Kuo-Fuchs	-	nul	-	+	-	++	++	+	maillage	+
ajout de dimension	+	nul	-	+	nul	-	--	--	*	--
Diogenes	+++	nul	+++		++	--	--	--	2-D	++
remplacements successifs (1)	++	nul	++	-	-	-	+	-	tous types	-
remplacements successifs (2)	+	nul	+	+	-	+	+	-	tous types	-
reconfiguration locale (1)	-	nul	--	++	+	++	-	++	tous types	++
reconfiguration locale (2)	+	++	-	-	+	--	+	+	2-D	+
reconfiguration locale (3)	++	nul	-	+	--	+	-	-	tous types	-

* : l'étude est à faire pour chaque type de réseau; elle existe actuellement pour des hypercubes et des maillages

Tableau 1 : récapitulatif des avantages et inconvénients des différentes solutions de reconfiguration du réseau suivant 5 critères :

- *la tolérance aux fautes :*
  - *(1a) taux de reconfiguration des processeurs ;*
  - *(1b) possibilités de reconfiguration des interrupteurs (switches) et interconnexions ;*
  - *(1c) le nombre de fautes tolérées à coup sûr (DFT : Deterministic Fault Tolerance).*
- *Le surplus matériel :*
  - *(2a) le surplus des interrupteurs (switches) et interconnexions en terme de surface ;*
  - *(2b) la modularité de la tolérance aux fautes concernant les processeurs ;*
  - *(2c) le surplus d'interconnexions en terme de bande passante ;*
  - *(2d) la facilité d'une implémentation physique.*
- *Les dégradations en performance du système (3) ;*
- *Le type de réseau d'interconnexion sur lequel la méthode peut s'appliquer (4) ;*
- *Et enfin, la simplicité et la vitesse de reconfiguration (5).*

La proximité physique entre les processeurs a souvent été abandonnée au profit du taux de reconfiguration comme on peut le voir dans le tableau comparatif. Or, pour les technologies les plus avancées, le temps de propagation dans les interconnexions devient prédominant. Il semble donc important de se rapprocher à nouveau du but de limitation de la longueur des interconnexions. D'autre part, la volonté d'atteindre un meilleur compromis entre le taux de reconfiguration et le surplus matériel requiert le choix du nombre de processeurs supplémentaires. Il apparaît donc nécessaire de concevoir une méthode de reconfiguration permettant d'avoir un nombre de processeurs supplémentaires parfaitement modulaire. Aucune des techniques présentées ci-dessus ne permet actuellement d'atteindre ces deux objectifs.

Par ailleurs, dans tous ces réseaux, le choix de l'architecture tolérante aux fautes n'est pas laissé au concepteur. Pire encore, la conception physique du réseau ne rentre pas en compte dans les critères permettant d'aboutir à la méthode de reconfiguration, ce qui entraîne les conséquences suivantes :

- pour certaines implémentations physiques des réseaux, aucune des méthodes ci-dessus ne peut être employée sans modification ;
- les pertes en performance dues au placement / routage peuvent être très importantes.

Enfin, ces méthodes ne permettent généralement pas d'optimiser le réseau en fonction de ses caractéristiques, comme par exemple l'utilisation de liens mono-directionnels plutôt que de liens bi-directionnels.

7 Conclusion

Nos avons présenté dans cette partie un survol des principales méthodes de reconfiguration existantes. Leur comparaison montre que celles-ci sont adaptées à différents cas et permettent souvent d'atteindre un très haut degré de fiabilité.

Par contre, il n'existe aucun schéma indépendant de l'implémentation physique de l'architecture cible, permettant un contrôle précis de la fiabilité par une modularité fine du nombre processeurs et minimisant les pertes en performance. Nous allons donc nous attacher à proposer un tel schéma de reconfiguration. Pour ce faire, nous nous plaçons dans le cadre particulier des structures régulières et, sans faire d'hypothèse sur la structure du réseau, nous en déduisons un nouveau schéma de reconfiguration.

Chapitre 5 : présentation de la méthode de reconfiguration

1 Introduction

Le chapitre précédent a permis de mettre en exergue les faiblesses des méthodes de reconfiguration actuelles vis-à-vis de notre besoin.

Des algorithmes de reconfiguration associés à des codes correcteurs d'erreur aux reconfigurations locales, ces méthodes impliquent toutes le choix du nombre d'interconnexions supplémentaires et/ou le choix du nombre de processeurs supplémentaires. Elles imposent donc le choix de la topologie de tolérance aux fautes de la structure.

Notre approche est inverse, puisque à partir d'un réseau tolérant aux fautes contraint par les entrées/sorties et le placement/routage des blocs le composant, nous allons appliquer la méthode de reconfiguration. L'avantage principal d'une telle approche est la liberté laissée au concepteur du degré de tolérance aux fautes souhaité. Nous allons donc chercher à rendre la méthode de reconfiguration la plus indépendante possible de l'implémentation physique de la tolérance aux fautes.

Afin d'atteindre ce but, nous allons séparer les deux grandes parties, habituellement fortement couplées, de la reconfiguration :

- le placement des processeurs sur le réseau physique ;
- le routage des interconnexions entre les processeurs ainsi placés.

Le routage, qui prend en compte les spécificités de chaque réseau, est traité ici sous la forme d'un exemple d'algorithme non optimal mais de durée acceptable, et ne constitue pas le cœur de notre technique de reconfiguration. Par contre, le placement des processeurs fait l'objet de notre plus grande attention. Nous chercherons à optimiser le placement afin que le routage puisse s'effectuer dans les meilleures conditions possibles. Différentes techniques sont également proposées afin de réduire le temps d'exécution de l'algorithme tout en maintenant un placement correct.

La méthode qui va maintenant être présentée ne dépend pas de notre architecture particulière de tolérance aux fautes, mais d'une classe plus large de réseaux qui va être détaillée par la suite. Aussi, nous avons souhaité que ce chapitre soit indépendant, car il sort du contexte des architectures SIMD ou multi-SIMD, ou même de celui des calculateurs parallèles.

2 Le contexte de l'étude

La reconfiguration est une solution de tolérance aux fautes qui s'impose pour certaines structures matérielles. C'est le cas, par exemple, de blocs de mémoire : ces blocs étant très nombreux, il est naturel qu'un certain nombre de ceux-ci soit ajouté et utilisé lorsqu'un bloc fonctionnel devient fautif. En règle générale, la reconfiguration est une solution « naturelle » pour les structures comportant un

grand nombre d'éléments identiques. Les mémoires en sont un exemple, mais nous pouvons également citer les réseaux de commutateurs, les FPGA (Field Programmable Gate Array) et, bien sûr, les calculateurs parallèles.

Toutefois, la reconfiguration entraîne un coût matériel et une complexité. Cette dernière peut alors rendre caduque l'utilisation de reconfiguration sur certaines structures. Par conséquent, nous ne pouvons appliquer la reconfiguration qu'à certaines structures que nous détaillons par la suite.

Les définitions que nous donnons ici font appel aux termes habituels de notre domaine d'application ainsi qu'aux notations de la théorie des graphes. Ainsi, nous employons indifféremment les termes élément, nœud ou sommet; arêtes, liens ou interconnexions; graphe ou structure (qui est, dans ce cas, un graphe non orienté). Nous ne supposons rien sur les éléments constituant les nœuds du graphe, qui pourront donc être aussi bien des processeurs, que des mémoires ou d'autres éléments complexes, à la seule condition que ces éléments soient tous identiques.

2.1 Structures régulières et notion de distance

La méthode de placement s'applique à toutes les structures reprenant les caractéristiques qui vont être présentées ci-dessous. Nous partons d'une classe de structures que nous appelons structures régulières.

Définition : une structure est dite régulière si chaque nœud interne la composant a le même degré d (même nombre d'arêtes), et si les nœuds composant la frontière de la structure (si cette frontière existe) ont un degré inférieur à d.

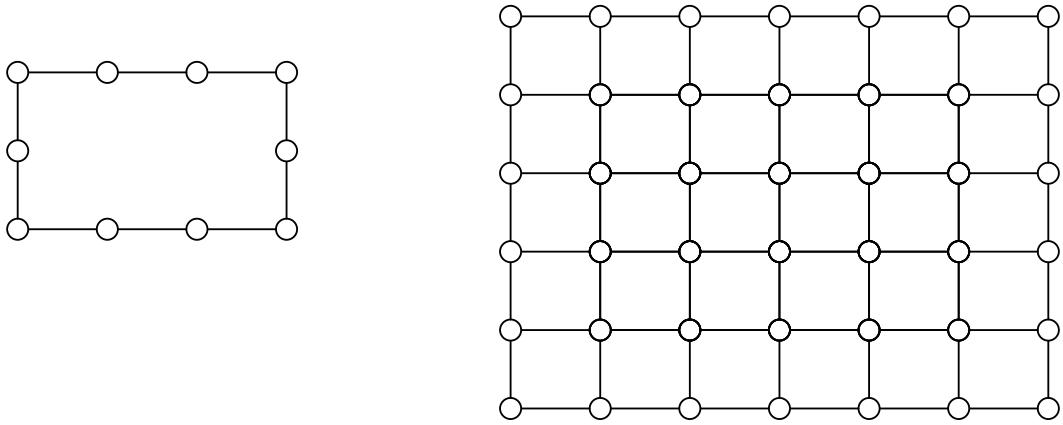
Cette définition diffère de celle couramment utilisée dans la théorie des graphes (un graphe régulier est un graphe dont tous les nœuds ont le même degré) car on tient compte ici de la frontière du graphe lorsqu'elle existe.

Nous pouvons citer à titre d'exemples de structures régulières :

- les anneaux (forme très simple) de degré 2 (figure 1a) ;
- les maillages 2-D ou grilles (figure 1b) de degré maximal 4 ;
- les tores 2-D de degré 4 ;
- les maillages 3-D de degré maximal de 6 ;
- les hypercubes à n dimensions de degré maximal n+1 ;
- les arbres
- les pyramides
- …

La classe des structures régulières englobe donc une grande partie des réseaux statiques classiques. A partir de cette première définition, il est possible de définir la notion de distance.

Définition : la distance entre deux nœuds d'une structure régulière est le nombre minimal de liens (arêtes) nécessaire pour aller d'un nœud à l'autre. C'est donc la longueur d'une chaîne minimale entre 2 nœuds.



a) anneau **b)** grille ou maillage 2-D

Figure 1 : exemples de structures régulières simples

2.2 Structures configurables et tolérantes aux fautes

Deux autres structures peuvent maintenant être déduites de la définition d'une structure régulière. Pour commencer, nous rappelons la définition d'un plongement de la théorie des graphes.

Définition : un plongement d'un graphe G dans le graphe H est défini par la donnée d'une application f de l'ensemble des nœuds de G dans l'ensemble des nœuds de H, et d'une application P_f de l'ensemble des arêtes de G dans l'ensemble des chaînes de H qui associe à chaque arête [x,y] de G une chaîne reliant les sommets f(x) et f(y) dans H.

Nous déduisons de cette définition celle d'une structure régulière configurable.

Définition : Etant donnée une structure régulière Sr comprenant n nœuds, une structure régulière configurable S_{rc} est définie par :

- un graphe non orienté composé de n éléments simples (nœuds simples), d'éléments dits d'intercommunication appelés également switches (nœuds d'interconnexions) et de liens entre nœuds ;
- une application *bijective* f, de l'ensemble des nœuds de S_r sur le sous-ensemble des nœuds simples de S_{rc} ;
- une application *bijective* P_f, de l'ensemble des arêtes de S_r dans un sous-ensemble des chaînes de S_{rc}.

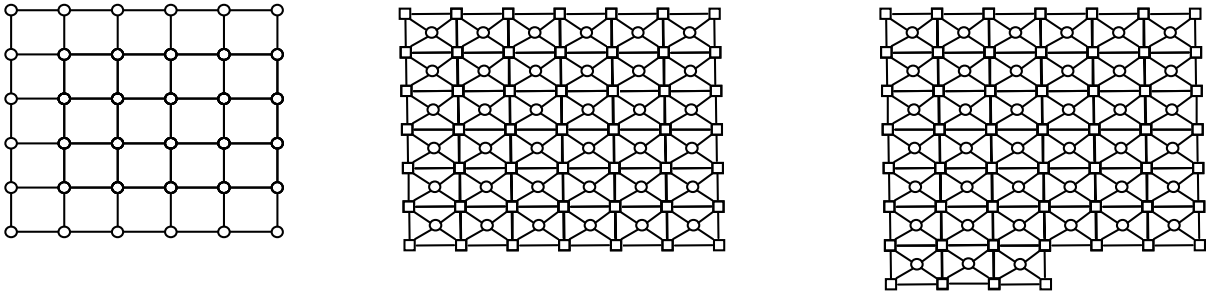
Une structure régulière configurable peut être utilisée pour la tolérance aux fautes.

Définition : une structure S_{rt} régulière pour la tolérance aux fautes (RTF) est définie par rapport à une structure régulière S_r comportant n nœuds de la façon suivante :

- une structure régulière configurable comportant p>n nœuds simples dont s = p-n nœuds sont appelés éléments supplémentaires ou spare ;
- un ensemble d'applications g_i bijectives de l'ensemble des nœuds n de S_r vers un sous-ensemble de n nœuds simples de S_{rt} ;
- pour chaque application g_i, une application P_{gi} bijective de l'ensemble des arêtes de S_r vers un sous-ensemble des chaînes de S_{rt}.

L'ensemble des applications g_i définit le ***placement*** des nœuds de S_r sur S_{rt} alors que pour chaque placement, l'application P_{gi} définit le ***routage*** de la structure S_{rt} par rapport à S_r. En terme de tolérance aux fautes, le placement tient compte des nœuds simples fautifs alors que le routage prend en compte les nœuds d'interconnexion et les liens fautifs.

La figure 2 montre un exemple des 3 types de structures de complexité croissante.



a) structure régulière **b)** structure configurable **c)** structure RTF

Figure 2 : les structures régulières à partir d'une grille 2-D

2.3 Distance, place logique et place physique

Les définitions de distance, de places logiques et physiques sont maintenant données. Ces définitions servent à introduire des notions qui sont utilisées dans le procédé de placement. On se donne une application g_{ref} qui constitue le placement des nœuds de S_r sur S_{rt} lorsque tous les nœuds peuvent être utilisés (pas de nœud simple fautif). La distance sera toujours prise par rapport aux nœuds simples dans la suite.

Définition : la distance entre deux nœuds simples d'une structure S_{rt} est égale au nombre de nœuds d'interconnexion compris dans une chaîne minimale reliant les deux nœuds.

Définition : la place logique d'un élément d'un réseau régulier S_r dans le réseau S_{rt} est la place que l'élément occupe lorsque la fonction de placement est g_{ref}.

Définition : la place physique d'un élément d'un réseau régulier S_r dans un réseau S_{rt} est la place que cet élément occupe dans la structure après application d'une fonction de placement g_i.

Ces définitions donnant le cadre général dans lequel le procédé peut être appliqué, il est maintenant possible de définir précisément les fonctions g_i et P_{gi}, c'est à dire les procédés de placement et de routage.

3 Le procédé de placement

3.1 Introduction

L'idée principale qui conditionne le procédé est la suivante : pour augmenter les chances de routage de la structure, il est important de placer correctement les éléments les uns par rapport aux autres, et notamment de réduire au maximum la distance entre des éléments voisins logiquement. La notion de distance ayant été définie au paragraphe précédent, le procédé devra donc répondre à la question suivante :

« Etant donnée une structure RTF, comment placer les éléments logiques sur la structure physique de façon à minimiser la somme des distances entre les places physiques des voisins logiques ? »

La méthode proposée pour résoudre cette question est tout d'abord présentée, puis nous présentons ses limites pour finalement apporter quelques propositions permettant de l'améliorer.

3.2 La méthode proposée

3.2.1 Introduction

La question du paragraphe précédent peut être décomposée en 3 questions qui seront traitées indépendemment :

1. comment choisir les places physiques des éléments de façon à minimiser la distance entre éléments voisins logiquement ?
2. comment assurer que le placement des processeurs logiques sur la structure physique est sûr quelles que soient les configurations de processeurs fautifs et quel que soit leur nombre (à condition, bien entendu, qu'il soit inférieur ou égal au nombre de processeurs supplémentaires) ?
3. comment assurer la cohérence du réseau, c'est à dire que dans chaque dimension, les éléments logiques ne soient pas inversés ?

3.2.2 Minimisation de la distance

La solution à cette première question consiste à essayer de placer les éléments les uns après les autres en respectant la règle suivante :

Règle 1 : la place de chaque élément doit être trouvée autour de la position de départ (place logique) en utilisant les éléments à une distance 0 puis 1, puis 2, puis 3... de la place logique de l'élément.

En effet, la place physique permettant à l'élément de se situer le plus près de ses voisins logiques est la place logique de l'élément. Si cette place est fautive ou occupée par un autre élément déjà placé, il faut rechercher une autre place. La structure étant régulière, il n'y a aucune direction à

privilégier, et la place la plus proche de ses voisins logiques est une des places logiques de ces voisins, donc une place située à une distance 1 de la place logique de l'élément. Si toutes les places à cette distance sont occupées ou fautives, les places suivantes sont celles situées à une distance 2, etc.

Cette règle nous donne une clé pour un bon placement, mais ne permet absolument pas de garantir le placement, condition énoncée par la question 2. De plus, selon l'ordre de placement et donc des placements précédents, on peut avoir un phénomène d'inversion dans une des dimensions. Ces deux problèmes sont maintenant traités.

3.2.3 Sûreté du placement

Pour répondre à la question 2 qui nous demande d'obtenir un algorithme de placement sûr à tous les coups, nous définissons tout d'abord un ordre de placement des éléments. Cet ordre sera obtenu par la définition d'un élément origine et d'une suite d'éléments englobant tous les éléments logiques du système. Nous verrons que cet ordre de placement influe fortement sur la qualité de l'algorithme de placement.

Nous définissons N comme étant le nombre total d'éléments à placer de la structure et S comme étant le nombre d'éléments supplémentaires à la structure. Alors, en suivant l'ordre donné ci-dessus, on définit S+1 places pouvant être utilisées pour le placement de chaque élément. Les places sont choisies de manière à satisfaire à la règle 1 et à la règle suivante :

Règle 2 : une fois défini un ensemble E de m places possibles pour les i-1 premiers éléments de la suite, l'ajout du $i^{ième}$ élément doit définir une et une seule place supplémentaire à l'ensemble E qui possède alors exactement m+1 éléments.

Démonstration : prenons l'ordre de placement défini. L'élément origine e_0 doit avoir S+1 places possibles pour assurer son placement dans le cas où S éléments seraient fautifs. A ce stade, l'ensemble E contient S+1 places (Card(E) = S+1).

L'élément suivant e_1 doit alors avoir au moins une place n'appartenant pas à E de façon à ce que l'ensemble des deux éléments (e_0 et e_1) ait alors S+2 places : soit au pire des cas, S places fautives et 2 places pour les deux éléments. En utilisant le même raisonnement, on peut donc déduire que chaque élément e_i doit avoir au minimum une place n'appartenant pas à E.

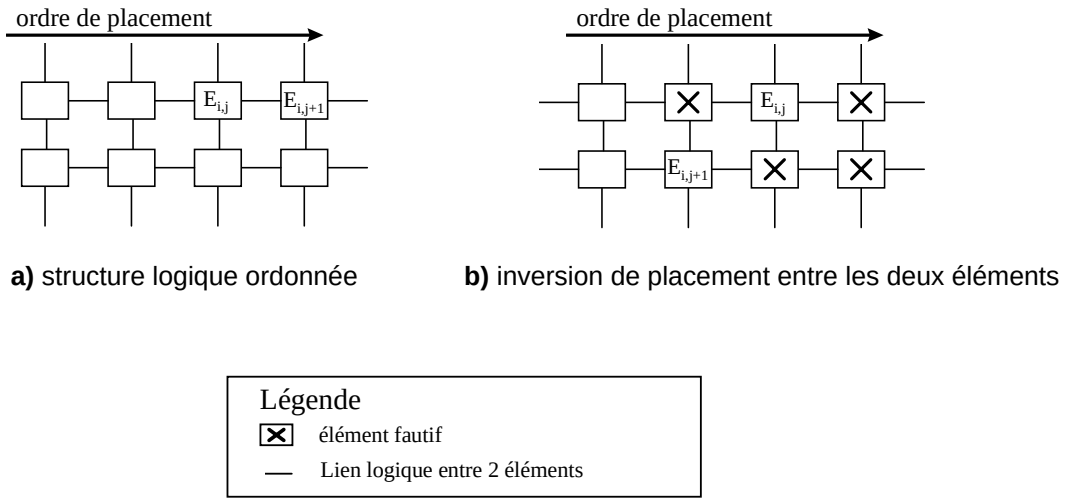
Calculons alors le nombre de places n'appartenant pas à E si tous les éléments de la suite n'ajoutent qu'une seule place à E.

Le nombre total de places disponibles est N+S. L'élément origine e_0 prend S+1 place et les N-2 éléments suivants prennent N-2 nouvelles places. Pour le dernier élément, il reste donc (N+S) - (S+1) - (N-2) = 1 nouvelle place possible. Il n'est donc possible d'ajouter qu'une et une seule place supplémentaire pour chaque élément.

Cette nouvelle règle restreint donc le placement des éléments. Une fois encore, la nécessité d'un choix judicieux de l'ordre de placement est mise en exergue.

3.2.4 Respect de cohérence

L'inversion de places logiques dans une dimension peut se produire comme il est montré figure 3. Cette inversion est à proscrire absolument, car elle entraîne des difficultés supplémentaires pour le routage, pouvant le rendre facilement impossible.



a) structure logique ordonnée **b)** inversion de placement entre les deux éléments

Figure 3 : exemple d'inversion de places logiques entre les deux éléments $e_{i,j}$ et $e_{i,j+1}$

Pour satisfaire à cette nécessité, on définit en fin de placement un mécanisme de respect de la cohérence qui consiste à détecter dans chaque dimension les inversions logiques par couple d'éléments voisins. Puis, on applique la règle 3.

Règle 3 : en cas de non-cohérence, les couples d'éléments impliqués sont inversés.

3.2.5 Conclusion

Les règles 1, 2 et 3 donnent la trame principale pour mener à un algorithme sûr de placement des éléments respectant une distance minimale entre les éléments (voir figure 4). Toutefois, les algorithmes de placement possibles restent nombreux et certains donneront de meilleurs résultats que d'autres. La principale inconnue est la détermination de l'ordre de placement. Cet ordre influe beaucoup sur l'efficacité du placement, et sa définition est nécessaire pour l'automatisation du procédé permettant d'améliorer le placement. La section suivante sera donc consacrée à la détermination de cet ordre.

3.3 Détermination de l'ordre de placement

3.3.1 Introduction

L'ordre de placement influe sur sa qualité. En effet, la règle 2 restreint les possibilités d'appliquer la règle 1, qui est la garante du respect de minimisation de la distance. Dans ces conditions, un ordre de placement défini correctement peut permettre de minimiser la restriction apportée par la règle 2. Dans cette optique, une solution de détermination automatique de cet ordre est proposée.

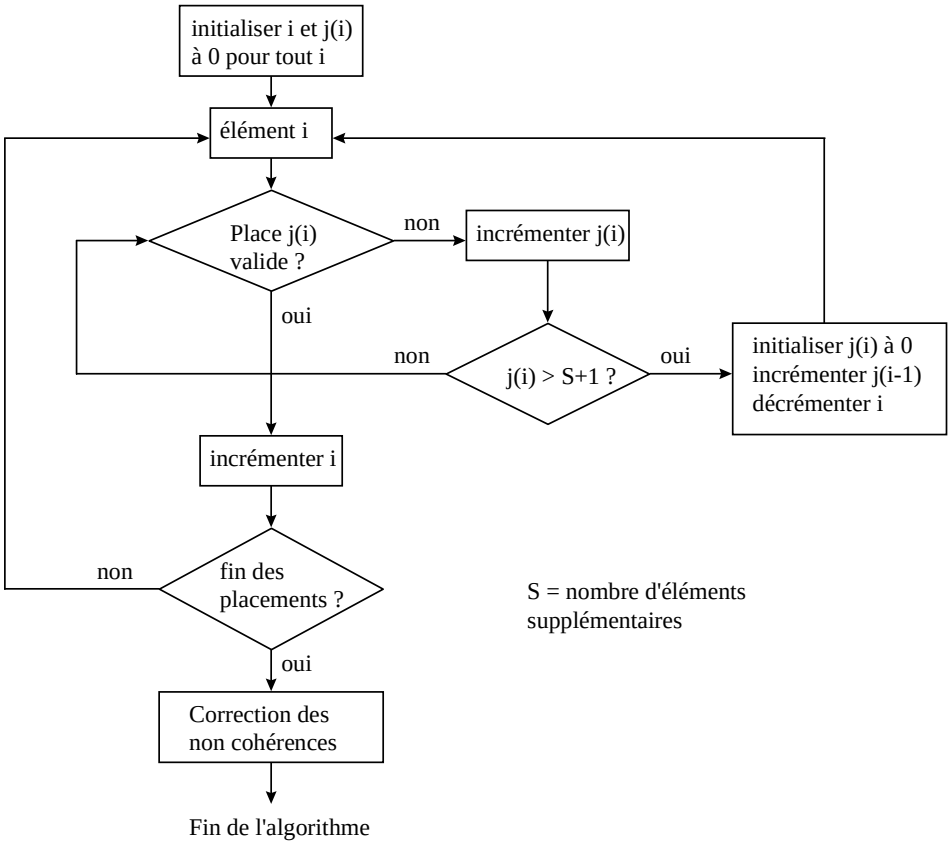


Figure 4 : synopsis de l'algorithme de placement des éléments déduit des règles 1,2 et 3 ; i représente l'i^{ième} élément logique à placer dans l'ordre de placement défini et j(i) est la j^{ième} place possible de cet élément i

3.3.2 Ordre de placement

Afin de respecter au mieux la règle 1, lorsqu'un nouvel élément est inséré, il doit posséder le plus de places possibles proches de sa place logique.

Cette remarque a deux conséquences :

- il est préférable de placer en premier un élément de degré minimal. Les directions suivant lesquelles un tel élément peut chercher une place possible sont réduites. De ce fait, cet élément est assuré de disposer d'une place proche de sa place logique et le nombre restreint de possibilités facilite le choix du prochain élément à placer ;
- il est préférable de placer en priorité les éléments dont les places proches appartiennent déjà à l'ensemble E des places possibles (défini pour la règle 2).

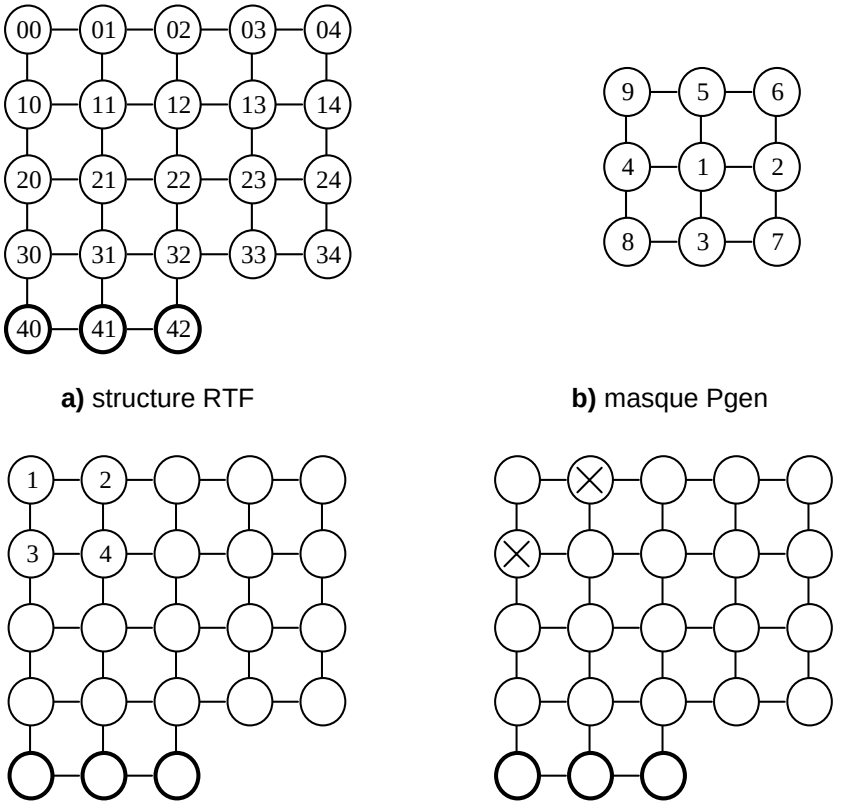
Nous pouvons alors définir la règle générale à respecter :

règle 4 : l'ordre de placement est défini par :

1. l'élément origine doit être choisi parmi les éléments du graphe ayant un degré minimal ;
2. on définit alors l'ensemble de ses places possibles que l'on ajoute à l'ensemble des places possibles E (préalablement vide) ;
3. la suite des éléments est définie en respectant la règle suivante : on choisit pour nouvel élément, un élément dont la place logique est dans E et dont le nombre de places voisines n'appartenant pas à E est minimal. A chaque étape, E est mis à jour.

Pour la définition des places possibles de chaque élément, nous utilisons un masque appelé Pgen. Ce masque dont le centre est la place logique de l'élément indique, par ordre de priorité, les places voisines acceptables de l'élément.

Afin d'étayer notre propos, prenons l'exemple de la structure logique de la figure 5a composée de 20 éléments et de 3 éléments supplémentaires marqués d'un trait gras. Le masque Pgen utilisé sera, pour cet exemple, celui de la figure 5b. Un choix judicieux du premier élément est alors l'élément nord-ouest (0,0). La figure 5c indique les places possibles pour ce premier élément et propose un ordre de priorité déterminé par le masque Pgen. La figure 5d désigne alors les deux éléments respectant la règle 4 qui peuvent être sélectionnés comme nouvel élément à insérer dans la suite de placement. Il est tout de même nécessaire d'ajouter un critère arbitraire pour le choix d'un élément lorsque plusieurs candidats sont disponibles. Ce critère peut, par exemple, favoriser le placement horizontal. Dans ce cas, l'élément (0,1) est l'élu. Et on réitère le processus. On rappelle, car c'est un point important que lorsque le masque est appliqué, on n'ajoute à E qu'une et une seule nouvelle place possible (règle 2).



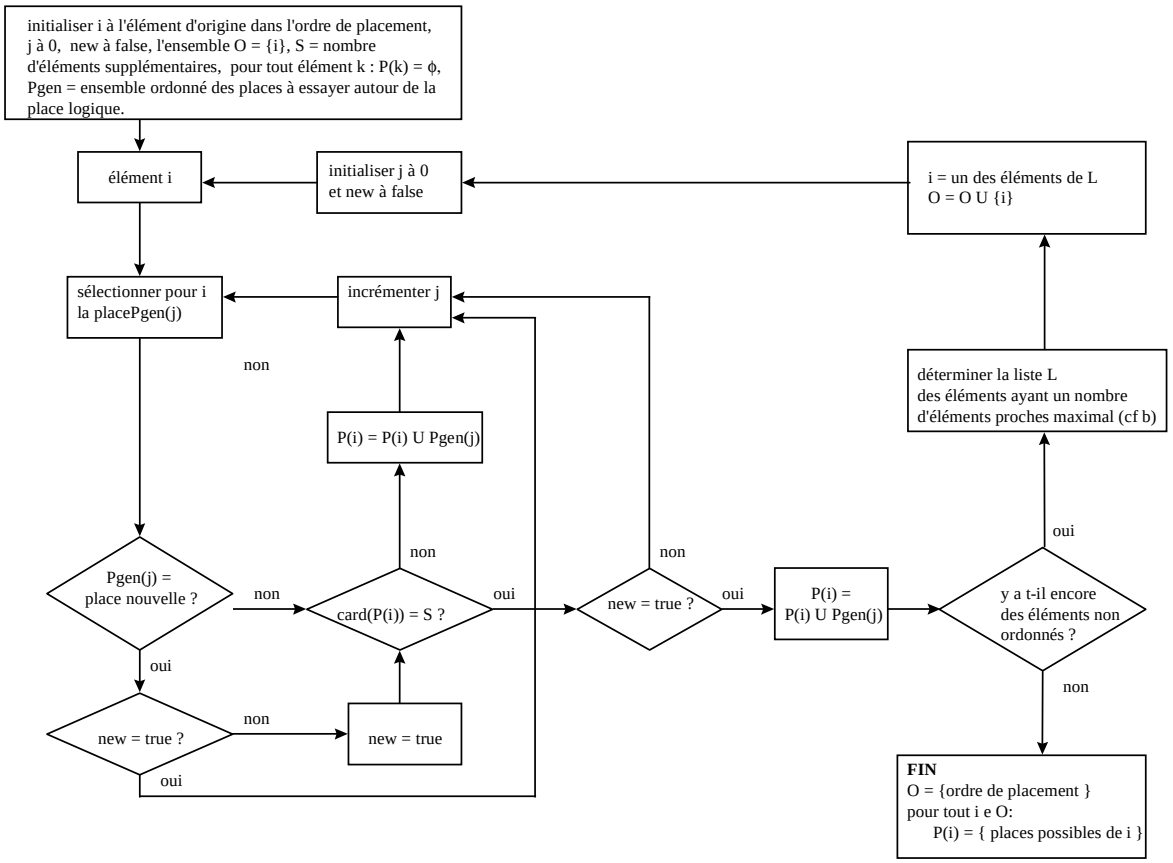
a) structure RTF **b)** masque Pgen

c) placement de l'élément (0,0) **d)** éléments suivants possibles

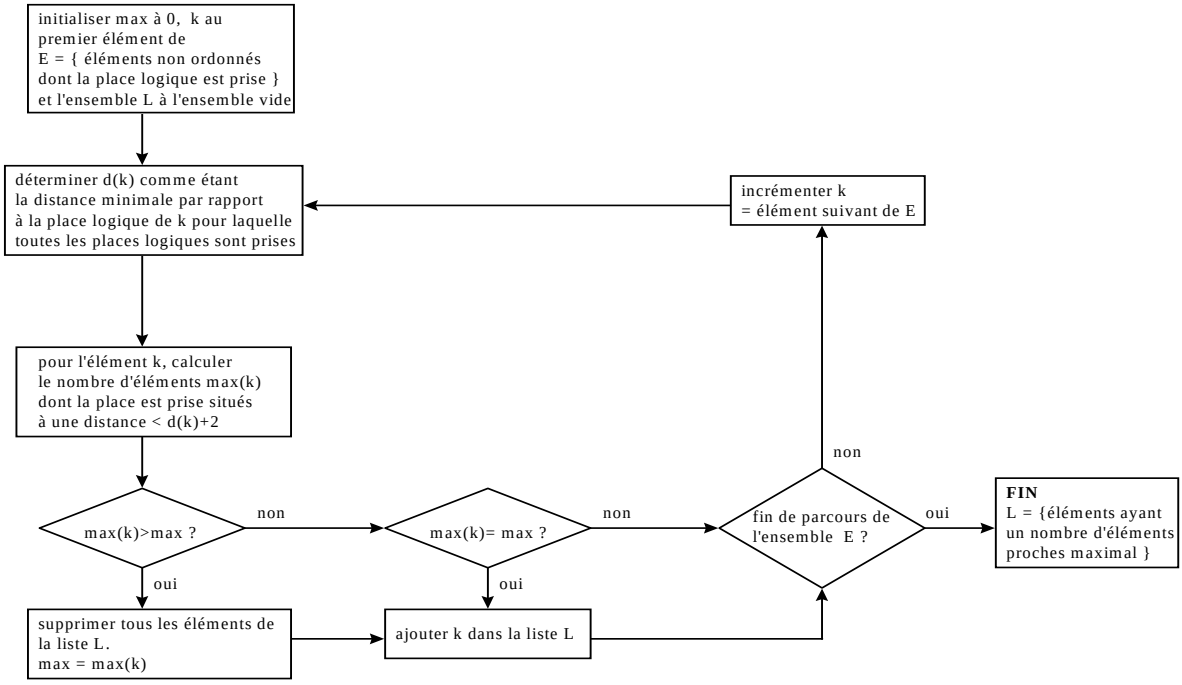
Figure 5 : exemple de détermination de l'ordre de placement et des places possibles de l'élément d'origine

3.3.3 Algorithmes

La détermination de l'ordre de placement et celle des places possibles d'un élément sont donc étroitement liés. La figure 6 donne l'algorithme global de détermination de ces deux éléments déduit de la section précédente. Cet algorithme permet d'obtenir à la fois la liste des éléments correspondant à l'ordre de placement et la liste ordonnée des places possibles de chaque élément. A partir des résultats de cet algorithme, le placement est effectué en respectant l'algorithme décrit figure 4. Il est important de noter que, pour une structure donnée, ce deuxième algorithme donnant l'ordre et les places de chaque élément n'est à dérouler qu'une seule fois, l'algorithme de la figure 4 étant le seul nécessaire lors de l'étape de reconfiguration du système.



a) algorithme complet de détermination de l'ordre de placement et des places possibles



b) détail de a) : obtention de la liste L à partir de l'ensemble E

Figure 6 : synopsis de l'algorithme servant à définir l'ordre de placement et les places possibles pour chaque élément

3.4 Réduction du temps d'exécution de l'algorithme : découpage en blocs

Dans le cas d'une structure comportant un nombre important d'éléments et/ou un nombre important d'éléments supplémentaires, la durée de l'algorithme décrit ci-dessus devient prohibitive. Il est donc nécessaire de proposer une meilleure solution.

Prenons l'exemple de la structure RTF de la figure 7a. Elle se compose de 16 éléments disposés suivant une matrice 4*4 et d'une ligne d'éléments supplémentaires. Si un algorithme permettait de placer chaque ligne indépendamment les unes des autres, la durée d'exécution de cet algorithme s'en trouverait diminuée. En règle générale, le découpage d'un réseau en un ensemble de blocs b_i et le placement de ces blocs de façon indépendante réduit la durée de l'algorithme.

Nous proposons maintenant une méthode qui garantit qu'il existe au moins une solution de placement. Nous définissons pour cela un ordre de placement comprenant un bloc d'origine b_0 et une suite de blocs b_i, puis la règle suivante :

règle 5 : afin de garantir le placement, chaque bloc doit respecter les règles 1,2 et 3 de la section 3.2 et les 2 points suivants :

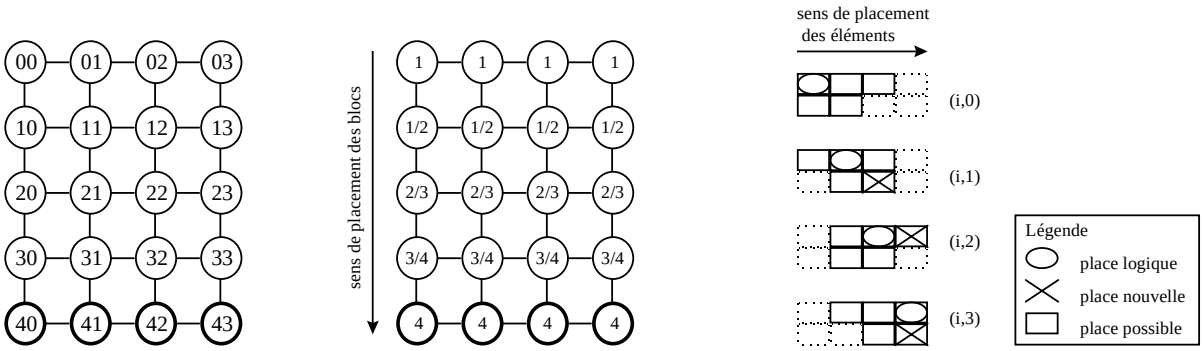
1. si S est le nombre d'éléments supplémentaires de la structure et M_i le nombre d'éléments du bloc b_i alors, ce bloc doit comporter exactement M_i-S éléments à placer et S éléments supplémentaires ;
2. pour tout bloc $b_i \neq b_0$, les M_i-S éléments à placer du bloc doivent être choisis parmi les éléments supplémentaires des blocs b_0 à b_{i-1}.

Démonstration : afin de tolérer S éléments fautifs, chaque bloc b_i doit comporter les N_i éléments à placer et exactement S éléments considérés comme éléments supplémentaires pour ce bloc, donc N_i + S éléments. D'autre part, comme la première place possible de chaque élément est sa place logique, d'après la règle 1, s'il n'y a aucune faute seules les places logiques des N_i éléments seront occupées. Par ailleurs, un seul élément de S sera utilisé pour couvrir une faute. De même, deux éléments de S seront utilisés pour couvrir deux fautes et ainsi de suite. Par conséquent, lorsqu'on cherche à placer les éléments d'un bloc b_i, il est possible qu'un nombre *nf* de places logiques aient été réquisitionnées par des éléments de blocs précédents pour couvrir un nombre *nf* de fautes. On utilisera alors *nf* places de S.

Le plus grand bloc étant le plus long à placer, sa taille détermine la durée de l'algorithme. Bien entendu, le choix des blocs est très important, car pour une meilleure efficacité de l'algorithme, ils doivent être de taille comparable. Pour des structures de type grille comportant une ligne d'éléments supplémentaires, la règle 5 est facilement applicable et nous reprenons tout de suite l'exemple de la figure 7a.

La figure 7b propose une décomposition en blocs efficace : des blocs de taille identique composés de deux lignes (une ligne d'éléments à placer et une ligne voisine d'éléments supplémentaires). Nous faisons le choix de placer les blocs depuis la première ligne jusqu'à l'avant-dernière (la dernière est formée de processeurs supplémentaires). On vérifie facilement que chaque bloc respecte bien la règle 5. La durée de l'algorithme est réduite à celle du placement d'une seule ligne. Par contre, les

placements sont moins bons, puisque certains voisins ne peuvent plus être utilisés. Néanmoins, cette solution simple peut s'avérer efficace dans de nombreux cas. Des blocs de taille plus importante peuvent également permettre de limiter les problèmes de respect de la règle 1, par exemple en intégrant 2 lignes d'éléments à placer et une ligne d'éléments supplémentaires.



a) structure RTF **b)** découpage en blocs **c)** algorithme de placement pour un bloc

Figure 7 : exemple de découpage en blocs

3.5 Réduction de la durée d'exécution des algorithmes

Les algorithmes présentés ci-dessus sont des algorithmes relativement simples tirés d'algorithmes de recherche en profondeur. Ce type d'algorithmes peut voir sa durée croître de manière exponentielle. En effet, la durée maximale de l'algorithme général est en $O(S^m)$ si m est le nombre d'éléments du réseau et S le nombre d'éléments supplémentaires et en $O(S^b)$ si b est le nombre d'éléments du plus grand bloc pour le placement par blocs. Pour de grands réseaux, il devient alors nécessaire de réduire b, ce qui risque, à terme, de diminuer notablement la qualité du placement.

Or, si on étudie en détail le placement des éléments, on se rend compte que le temps d'exécution de l'algorithme sera d'autant plus court que les éléments fautifs sont prêts des premiers éléments placés. Puisque les éléments fautifs et leur position sont connus avant le placement, il est possible de choisir l'élément d'origine e_0 ou b_0 et l'ordre de placement, de façon à minimiser la durée de l'algorithme.

Nous utiliserons toutefois une approche plus systématique et plus simple qui consiste à utiliser plusieurs placements définis par différents masques Pgen, différents éléments d'origine et différentes tailles de bloc, chacun de ces placements étant considéré comme non valide si :

1. le routage est impossible avec ce placement ;
2. le placement n'est pas effectué dans un temps déterminé.

Ainsi, la durée de l'algorithme devient déterministe. Nous verrons sur l'exemple de l'architecture parallèle proposée qu'une stratégie comme celle-ci conduit à de très bons résultats de placement pour

une durée d'algorithme très réduit, ceci même pour des réseaux de taille assez importante (400 processeurs).

Notons, enfin, que les schémas de reconfiguration comportant un faible nombre de processeurs supplémentaires, comme il en est proposé au chapitre 6, ne posent pas de problème de durée de l'algorithme qui dans ce cas, sera très simple et pourra être implémenté de façon matérielle.

3.6 Conclusion

Deux techniques de détermination de l'algorithme de placement des éléments ont été présentées. La première est une méthode qui découle directement des 3 règles énoncées dans la section 3.2 et qui permet de réaliser un placement « idéal » dans le respect de nos contraintes. Cependant, son coût en temps peut s'avérer prohibitif. La seconde technique consiste à découper le système en blocs « indépendants » et donne un placement moins bon, mais néanmoins suffisant dans la plupart des cas. La durée de cet algorithme est largement inférieure à celle de la première méthode. Il faut également noter que cette solution permet un contrôle du rapport entre la qualité du placement et la durée de l'algorithme : des blocs de taille importante donneront de meilleurs résultats de placement que des blocs de taille réduite aux dépens de la durée de l'algorithme. Cette dernière technique peut être utilisée efficacement en combinant les placements, comme nous le montrons au chapitre 6.

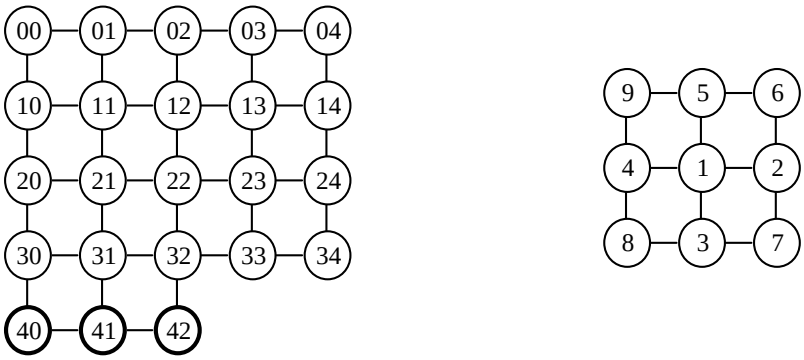
Tous les outils nécessaires pour le placement sont maintenant disponibles. Il est donc possible d'en montrer plusieurs exemples.

4 Exemples de placements sur un maillage 2-D

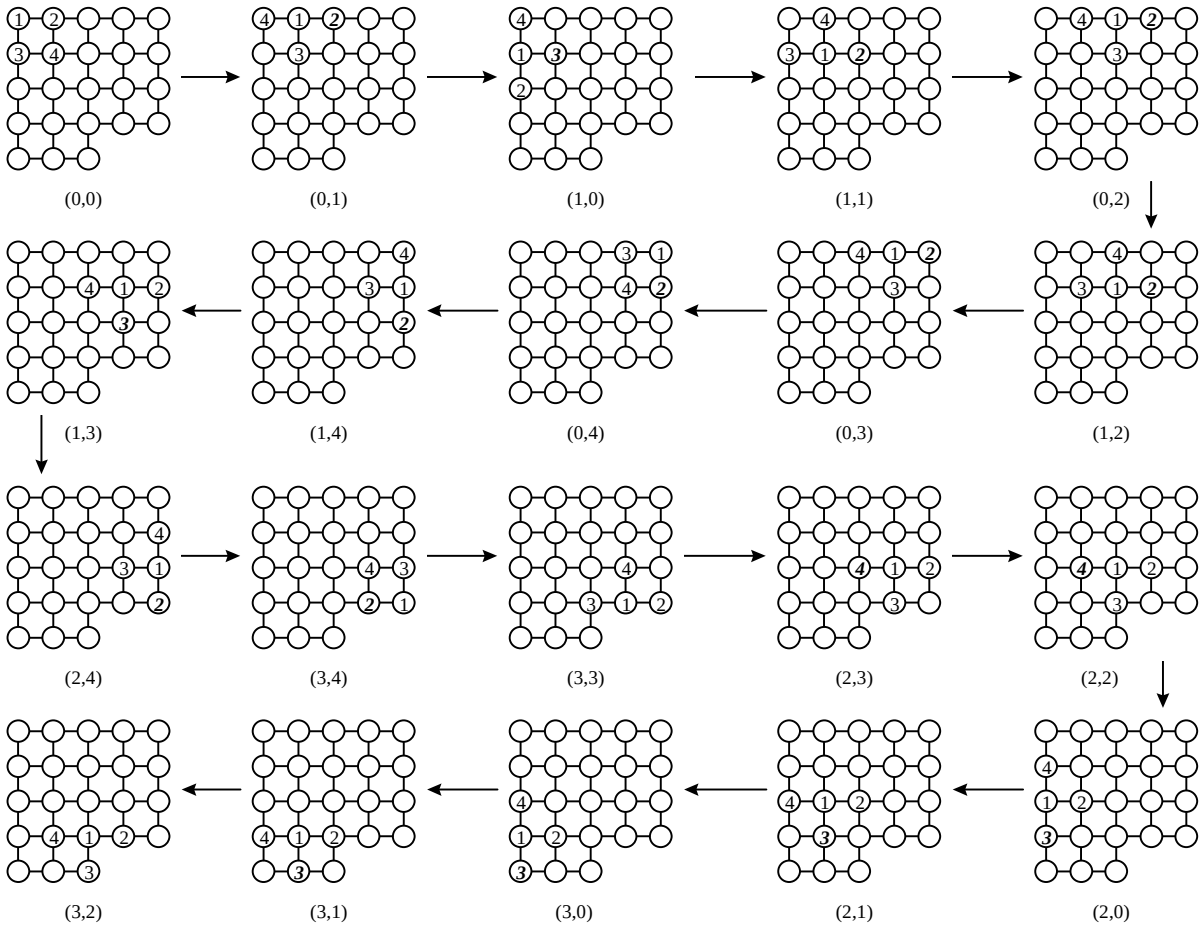
4.1 Placement complet

Nous prenons l'exemple de la figure 8a pour proposer un exemple de placement. Celui-ci utilise les règles 1,2,3 et 4 sans utiliser la technique de découpage en blocs. Comme il a été vu dans la section précédente, il faut dans un premier temps déterminer l'ordre de placement O des éléments, et, pour chaque élément i, l'ordre des places possibles P(i). L'élément origine est positionné en (0,0) selon le critère de la règle 4.

Il faut ensuite déterminer les places possibles et leur ordre de placement. Pour cela, nous proposons figure 8a un exemple de masque Pgen. Dans notre cas, ce masque, composé des éléments à une distance 1 et d'éléments à une distance 2, est suffisant pour permettre de placer tous les éléments. Les places possibles de l'élément origine sont alors déterminées en suivant l'algorithme de la figure 6a et la liste L des éléments suivants possibles par l'algorithme de la figure 6b.



a) structure RTF et masque Pgen



b) déroulement de l'algorithme de détermination de l'ordre de placement et des places successives de chaque élément

Figure 8 : exemple d'algorithme de placement

A partir des places possibles de l'élément d'origine, on obtient L = {(0,1),(1,0)}. Fixons comme critère de choix la priorité à la direction horizontale. C'est donc l'élément (0,1) que nous traitons ensuite. L'ordre de placement O devient alors O = {(0,0),(0,1)}.

Traitons l'élément (0,1). Suivant le masque Pgen, l'ensemble des places possibles se réduit au singleton {(1,0)}. Cet élément est donc ajouté à O qui devient O = {(0,0),(0,1),(1,0)}, et ainsi de suite.

Le déroulement de l'algorithme de détermination de l'ordre de placement est représenté figure 8b.

Il est à noter que les variables du placement sont :

- le choix de l'origine ;
- le critère de sélection d'un nouvel élément à placer lorsque la liste L en comporte plusieurs ;
- le choix du masque Pgen.

4.2 Placement par blocs

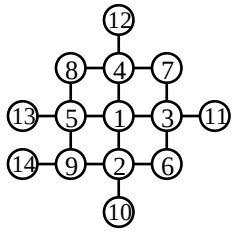
Un exemple très simple de placement pour une structure de grille, basé sur des blocs de deux lignes d'éléments a déjà été donné figure 7c. Un autre exemple est maintenant présenté sur un placement par blocs de 3 lignes, 2 lignes d'éléments N_i et une ligne d'éléments supplémentaires.

Afin de définir la qualité d'un découpage par blocs pour ces structures, nous utilisons deux critères :

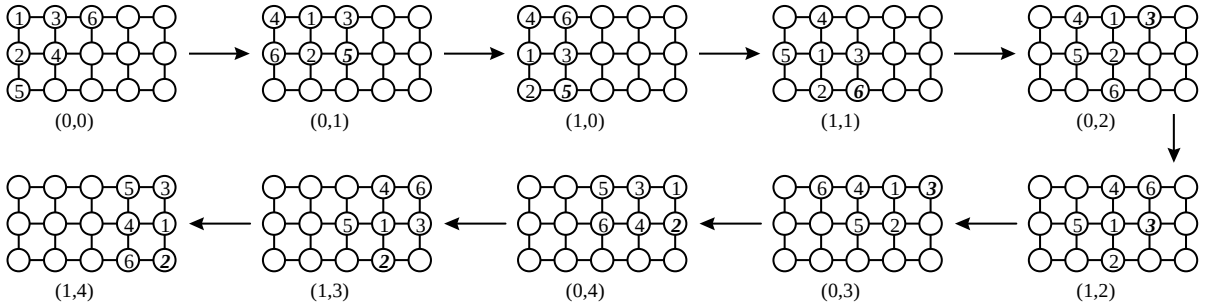
- le nombre n de places, voisines de la place logique, qui peuvent être utilisées sans discontinuité, malgré le découpage. En d'autres termes, pour une place logique, on considère tous les éléments disponibles à une distance i, en commençant bien évidemment par i=1. Si tous ces éléments sont dans le même bloc que la place logique, on réitère pour une distance i+1, et ainsi de suite. Dès qu'un des éléments considérés n'appartient pas au bloc, le processus est stoppé. Notons D la distance traitée à cet instant. Le nombre n correspond alors au nombre d'éléments accessibles dans le bloc pour une distance inférieure ou égale à D. Le placement sera alors dit *idéal pour n éléments*.
- le nombre de places appartenant au même bloc que celui de la place logique, et situées à une distance inférieure à trois de celle-ci. Nous dirons alors que le critère de distance à 3 est atteint pour s éléments.

Il est facile de se rendre compte qu'un découpage par blocs de 2 lignes peut être rapidement limité. En effet, le découpage n'est idéal que pour 3 éléments à l'intérieur du bloc, les voisins gauche, droite et bas, et 2 éléments aux extrémités du bloc. Par ailleurs, le critère de distance 3 est atteint pour 7 éléments à l'intérieur du bloc, et 4 à ces extrémités.

Par comparaison, le placement par blocs de 3 lignes présente des qualités nettement supérieures pour la deuxième ligne du bloc. Pour celle-ci, le découpage est idéal pour 10 éléments à l'intérieur de bloc et 6 éléments aux extrémités de la ligne, valeurs pour lesquelles le critère de distance 3 est atteint. Le placement de la deuxième ligne d'un bloc de 3 lignes aura donc tendance à « corriger » les erreurs de placement de la première ligne, améliorant considérablement la qualité de l'algorithme. Afin d'étayer cela, un exemple de déroulement de l'algorithme de détermination de l'ordre de placement pour un bloc de 3 lignes d'un réseau de n* 5 éléments est présenté figure 9.



a) masque Pgen utilisé



b) déroulement de l'algorithme de placement par blocs de deux lignes

Figure 9 : exemple d'algorithme de détermination de l'ordre de placement pour un bloc. En italique, la place nouvelle de chaque élément

5 Le procédé de routage

5.1 Introduction

Comme il a été dit dans l'introduction, cette partie ne fait pas vraiment l'objet du cœur du procédé de reconfiguration. Toutefois, il semble important de donner quelques indices quant à la réalisation d'un algorithme de routage. Nous considérons ici que la structure RTF est connue et parfaitement décrite par un graphe dont les nœuds sont les éléments et les éléments de commutation, et les arcs (ou arêtes) sont les interconnexions. Dans la suite, nous ne nous intéressons pas à un inventaire exhaustif des très nombreuses méthodes, mais nous nous attachons à un type d'algorithme très simple en relation avec notre problématique.

5.2 Algorithme de routage à chemins limités

Un des problèmes mis en exergue au chapitre précédent est l'absence de garde-fous pour limiter la longueur des chaînes (ou chemins) entre voisins logiques. Limiter la longueur des chemins entraîne forcément une baisse du taux de reconfiguration globale, mais permet également de limiter les effets désastreux d'un surdimensionnement du temps de communication nécessaire pour tenir compte de tous les cas de reconfiguration possibles. Ainsi, dans notre cas, lorsque le système ne peut pas se reconfigurer avec des chemins de taille réduite, il sera considéré comme n'étant plus fonctionnel, et donc en panne. Il s’agit donc d’un compromis entre le taux de reconfiguration et les pertes en performance induites.

Nous souhaitons donc limiter la taille des chemins avant d'appliquer l'algorithme de routage. Pour cela, nous limitons le nombre d'éléments d'interconnexion entre éléments logiquement voisins à une valeur *Imax*. La théorie des graphes fournit des algorithmes permettant de déterminer les chemins ayant une taille limitée. Ce sont les algorithmes de recherche des plus courts chemins. Un tel algorithme sera utilisé pour déterminer tous les chemins entre deux éléments, de taille inférieure à Imax avant d'utiliser l'algorithme de routage très simple suivant :

Initialiser i et j à 0.

***Point 0** : Pour le couple C_j d'éléments voisins logiques:*

***Point 1** : Si le chemin $C_j(i)$ existe :*

essayer le routage de C_j avec ce chemin

S'il y a conflit sur un élément de commutation

incrémenter i et retour au Point 1

Sinon,

S'il existe d'autres couples C d'éléments :

incrémenter j et retour au Point 0

Sinon,

Routage réussi, fin.

Sinon,

S'il existe un couple C_k de processeurs dont le routage rend le routage de C_j impossible

Supprimer tous les routages entre C_k et C_j

incrémenter le chemin i de C_k et retour au Point 0

Sinon

Routage impossible

Fin

Cette solution est également intéressante si on cherche à tolérer des fautes pouvant survenir dans les interconnexions ou les éléments d'interconnexion. En effet, dans ce cas, l'algorithme précédent peut être complété en prenant en compte les points suivants :

1. si une interconnexion déterminée comme fautive est reliée à un élément, l'élément est considéré comme fautif.
2. si une interconnexion déterminée comme fautive n'est reliée à aucun élément, tous les chemins empruntant cette interconnexion sont supprimés de la liste des chemins à essayer.
3. si un élément d'interconnexion est fautif :

- si une faute affecte une transmission d'un processeur vers un lien supplémentaire, le processeur est considéré comme fautif ;
- si une faute affecte une liaison entre deux liens supplémentaires, ces deux liens sont considérés comme fautifs ;
- si une faute affecte le contenu d'un registre de programmation d'un multiplexeur composant une partie de l'élément de commutation, tous les liens correspondants à ce multiplexeur seront considérés comme fautifs ;
- enfin, une faute affectant le système de programmation des éléments de commutation sera considérée comme fatale. Cette structure devra donc être conçue avec un soin tout particulier.

La difficulté, dans ce cas, ne réside pas dans la mise en place d'un algorithme tolérant les fautes du réseau, mais dans la détermination et surtout la localisation de ces fautes, problème qui est délicat dans de nombreux systèmes. Un test des interconnexions entre éléments permettant de faire une analyse des interconnexions fautives devra ainsi être mis en place. Il est à noter dans ce cas que pour des éléments électroniques équipés de cellules Boundary Scan avec la norme 1149.1, le test des interconnexions et leur analyse sont facilités.

6 Conclusion

Une technique de reconfiguration d'éléments dans une structure régulière a été présentée. Cette technique se veut le plus possible indépendante de l'implémentation physique du réseau RTF. Ainsi, le placement des éléments sur la structure se fait sans tenir compte, et sans influencer les interconnexions de la structure. Seul le routage tient compte de la topologie d'interconnexion utilisée, mais cette partie de la reconfiguration est minimisée. Cela entraîne un besoin de qualité maximal pour le placement des éléments. Pour cela, nous utilisons un procédé de minimisation de la distance globale de tous les éléments voisins logiques. Ce procédé est donc rendu indépendant de l'implémentation physique de la structure reconfigurable et ne dépend plus que de la structure logique. Mais il est également rendu indépendant de la technique de routage qui peut donc prendre différentes formes. Une de ces formes est donnée dans le paragraphe précédent en s'appuyant sur un critère de minimisation des chemins, et donc sur des algorithmes de la théorie des graphes classique de recherche du plus court chemin et de recherche en profondeur.

Cette technique de reconfiguration n'est pas limitée au cas des calculateurs parallèles, ni à celui de la tolérance aux fautes mais peut être utilisée pour tout système possédant une structure régulière logique ou physique et réclamant une optimisation du placement des éléments.

Comme il a été précisé dans l'introduction de ce chapitre, le fait que le procédé de reconfiguration soit indépendant de la conception du réseau reconfigurable entraîne la nécessité de traiter de l'optimisation de l'architecture. Aussi, le chapitre suivant propose une architecture reconfigurable en concordance avec notre approche.

Chapitre 6 : architectures globale et locale de réseau tolérant aux fautes

1 Introduction

Nous avons présenté, au chapitre précédent, un concept de reconfiguration indépendant de l'architecture cible. Cette technique est applicable à n'importe quelle structure régulière « adaptée » à la tolérance aux fautes et est complètement modulaire en termes de processeurs supplémentaires. Un de nos objectifs de tolérance aux fautes est donc atteint : la structure sur laquelle s'appuie la reconfiguration n'est pas imposée. Conséquence immédiate, l'architecture aura une grande incidence sur l'amélioration de la fiabilité.

L'architecture régulière considérée dans ce chapitre est de type maillage ou tore 2-D car le degré de ces réseaux est compatible avec les limitations en entrées et sorties des architectures à haut niveau d'intégration.

Le reste de ce chapitre est organisé ainsi : dans un premier temps, les concepts de l'architecture sont explorés et les choix sont justifiés. L'architecture générale est alors décrite en détail ainsi que quelques architectures déduites de celle-ci. Puis, nous décrivons quelques schémas de tolérances aux fautes possibles à partir des éléments donnés précédemment. Pour finir, nous donnons les résultats de taux de reconfiguration, de coût matériel et de fiabilité obtenus.

2 Concepts de l'architecture

2.1 Introduction

Le concept d'architecture fortement intégrée conduit à considérer le système comme composé de blocs élémentaires (BE). En effet, certains niveaux d'intégration constituent une limite technologique, car leur bande passante est plus limitée que les niveaux inférieurs. Un ASIC, par exemple, est un BE dont le nombre d'entrées/sorties est limité : on ne pourra donc ajouter que très peu d'interconnexions supplémentaires à l'interface du BE. Par conséquent, il paraît logique de séparer les choix d'architecture tolérante aux fautes en un choix global concernant l'ensemble des blocs élémentaires, et un choix local concernant l'intérieur de chaque bloc élémentaire. Par ailleurs, deux niveaux de tolérance aux fautes permettent d'obtenir une meilleure fiabilité. Ainsi, en cas de défaillance d'un des systèmes de sécurité, le deuxième peut prendre le relais.

Nous avons donc mis en place deux niveaux de tolérance aux fautes qui sont tout à fait complémentaires et peuvent être combinés ou utilisés séparément. La suite de cette section sert à présenter ceux-ci.

2.2 Choix local

2.2.1 Introduction

Nous utilisons, pour la définition de l'architecture, les contraintes sur les blocs élémentaires :

- Le surplus matériel en terme de processeurs supplémentaires doit être parfaitement modulable, ceci afin de pouvoir limiter ce surplus en fonction de la fiabilité désirée ;
- la structure ne doit faire intervenir aucune interconnexion supplémentaire avec l'extérieur du bloc élémentaire ;
- les interconnexions internes supplémentaires doivent être limitées.

La première contrainte implique que la reconfiguration doit être à grain fin et modulaire. La deuxième contrainte mène à une reconfiguration locale alors que la troisième demande une méthode de routage qui puisse être adaptée à plusieurs implémentations de la même architecture. La méthode de reconfiguration présentée au chapitre précédent répond parfaitement à tous ces critères.

Nous pouvons alors proposer une architecture non corrélée avec la méthode de reconfiguration. Pour cela, 3 éléments qui formeront l'architecture tolérante aux fautes sont définis : les liens ou interconnexions supplémentaires, les éléments de commutation et les processeurs spare. Le choix de chacun de ces éléments est maintenant discuté.

2.2.2 Interconnexions

Dans le cas des calculateurs fortement intégrés à réseau en matrice (voir figure 1), il est totalement irréaliste d'augmenter de manière trop importante les interconnexions internes. En effet, celles-ci risquent de rendre la réalisation de la structure physique difficile, notamment si le bloc élémentaire est un ASIC.

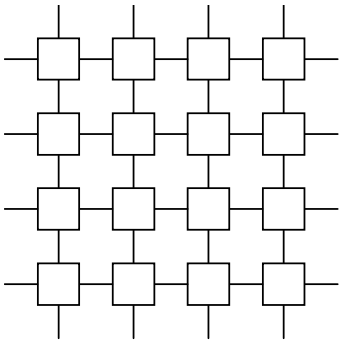


Figure 1 : exemple d'un sous-réseau 4*4 d'un maillage 2-D (grille)

Toutefois, il est évident que nous ne pouvons pas obtenir de bons taux de reconfiguration sans un certain surplus d'interconnexions : pour la simple dérivation d'une ligne ou d'une colonne, celui-ci est déjà de 50%. Lorsqu'on veut combiner une reconfiguration dans les deux dimensions, le surplus dépasse alors forcément les 100%. Ainsi, les réseaux tolérants aux fautes à grain fin décrits dans l'état de l'art (chapitre 4) comportent au minimum 200 % d'interconnexions supplémentaires. Les surplus

d'interconnexions de ces approches nous donnent une indication du minimum de surplus permettant d'obtenir une bonne reconfiguration. Nous nous baserons donc sur le surplus minimum de 200% des interconnexions internes pour chaque bloc élémentaire. Nous verrons par la suite que ce surplus peut être encore limité dans certains cas de figure.

La minimisation du surplus d'interconnexions requiert une attention toute particulière vis-à-vis de l'architecture, c'est à dire au placement des interconnexions et, surtout, à la structure des éléments d'interconnexions.

2.2.3 Eléments d'interconnexion

Les éléments d'interconnexion sont également appelés switches dans la littérature. Ces deux termes seront repris indifféremment par la suite. Ces éléments servent à réaliser la liaison physique entre les interconnexions. Ils font donc une commutation de circuit dans la structure.

Les switches sont en général assez simples et minimisés. Néanmoins, les possibilités technologiques actuelles ouvrent de nouvelles possibilités. En effet, le nombre de couches de métaux disponibles devient très important et le nombre de transistors que l'on peut utiliser pour ces circuits peut être plus conséquent. Il est ainsi envisageable d'obtenir des switches très complexes à temps de traversée réduit.

Afin de simplifier le routage des interconnexions lors de la conception de la structure, nous allons chercher à limiter le nombre de switches. Ceci doit permettre de limiter la longueur des interconnexions physiques, et donc d'améliorer la fréquence de la structure. Nous préférons ainsi disposer de switches peu nombreux, même s'ils sont de taille plus conséquente.

Le réseau 2-track n-spare (voir chapitre 4) est le réseau qui répond le mieux aux critères énoncés ci-dessus. Celui-ci propose l'utilisation de deux types de switches, les PT (Processor-to-Track) et les TT (Track-to-Track). Les premiers servent à relier les processeurs au réseau des interconnexions supplémentaires, alors que les seconds servent à relier entre elles ces interconnexions (figure 2). En partant de ce réseau, afin d'atteindre notre objectif, nous allons chercher à réduire le nombre d'éléments d'interconnexions. Dans un premier temps, nous les réduisons de 8 à 4. Cela est possible par une fusion des switches TT et PT pour obtenir un seul switch plus complexe (figure 3). Cette fusion entraîne donc une augmentation de la complexité des switches, mais augmente également les possibilités de commutation. Ainsi, la figure 4 montre un exemple de routage possible avec les switches regroupés, impossible avec la configuration 2-tracks n-spare correspondante.

Une fusion plus importante des switches est encore possible. Elle est d'ailleurs réalisée pour améliorer les communications dans de récents supercalculateurs [Boku1997]. La solution proposée est de fusionner les switches dans les lignes et les colonnes pour obtenir des « barres » d'intercommunications de type crossbar avec la possibilité de passer d'un niveau à un autre (changement ligne-colonne) à chaque nœud (voir figure 5). Cette solution, très intéressante du point de vue des communications, est toutefois très lourde et complexe à mettre en oeuvre et nécessite de véritables nœuds de calcul pour permettre les communications. De plus, ce schéma n'est pas directement modulaire et il est nécessaire de concevoir de nouveaux éléments d'interconnexion pour

chaque taille de réseau. Cette structure ne peut donc convenir à des calculateurs parallèles dont la taille est réduite au minimum.

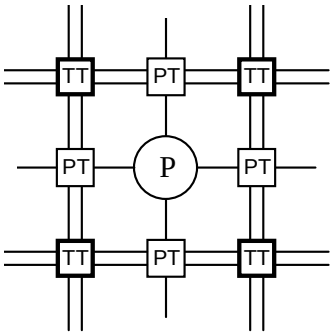


Figure 2 : détail des interconnexions d'un réseau 2-track n-spare. Deux types de switches sont utilisés, les « Track to Track » (TT) et les « Processor-to-Track » (PT)

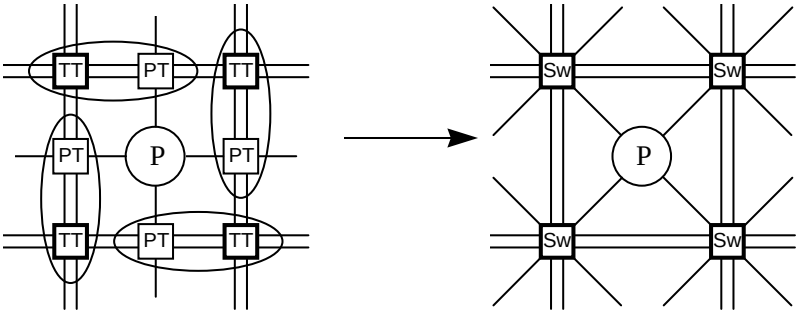


Figure 3 : transformation du réseau 2-track n-spare par diminution du nombre de switches

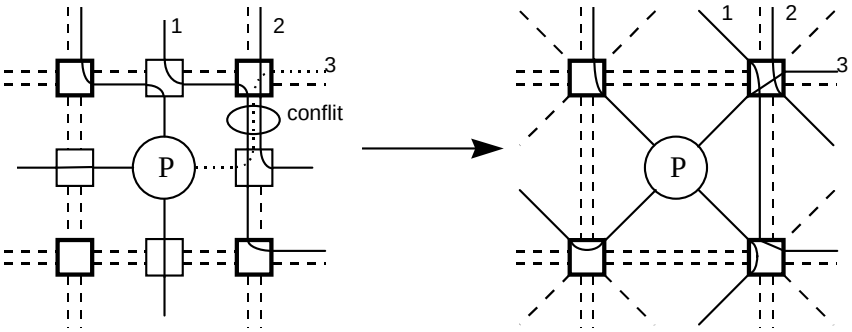


Figure 4 : exemple de conflit résolu avec la nouvelle structure

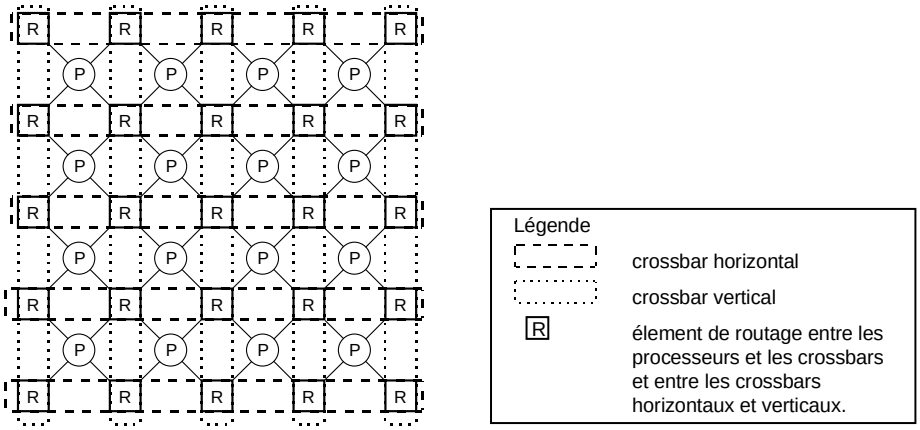


Figure 5 : réseau formé de crossbars horizontaux et verticaux

A partir de la définition des switches, différents choix architecturaux sont possibles, notamment au niveau du placement des interconnexions supplémentaires entre les switches. Toutefois, il est important de pouvoir minimiser la longueur des interconnexions entre les processeurs. Pour cette raison, il est plus facile de maîtriser le placement/routage (au sens de la réalisation physique) du circuit lorsque ces interconnexions sont parfaitement régulières et qu'elles ne se croisent pas. Ainsi, et afin de faciliter le placement/routage du bloc élémentaire, nous nous limiterons à une architecture très simple.

2.2.4 Processeurs supplémentaires

Le choix du nombre de processeurs supplémentaires dépend du compromis entre le surplus matériel qu'on s'accorde pour la tolérance aux fautes et le besoin en fiabilité. Ainsi, il est possible que les calculs de fiabilité mènent à ne vouloir tolérer qu'une seule faute par bloc élémentaire.

Mais les demandes peuvent également être plus importantes et consister en une amélioration conséquente de la fiabilité du système qui se traduit par une augmentation importante de la fiabilité de chaque bloc élémentaire. En effet, sur une structure fortement intégrée et embarquée, il est utopique de penser pouvoir réparer le système par une intervention extérieure. Dans ce cas, il est nécessaire de garantir une durée de fonctionnement importante, ou, pour être plus précis, une forte probabilité de l'atteindre celle-ci : on ne peut en effet parler que de probabilité en terme de durée de fonctionnement d'un composant électronique. Il est donc nécessaire de pouvoir choisir le nombre de processeurs supplémentaires par rapport à la fiabilité voulue. C'est pourquoi la structure choisie aura, contrairement à de nombreuses approches présentées dans le chapitre 4, un nombre modulaire de processeurs supplémentaires. Nous reviendrons sur le calcul de la fiabilité à la fin de ce chapitre afin de donner les outils pour déterminer le nombre de processeurs supplémentaires en fonction des besoins en fiabilité.

2.2.5 Conclusion

Des trois points précédents, nous pouvons donc déduire une architecture globale qui respecte nos objectifs. Cette architecture est présentée figure 6. On note que les entrées et sorties du réseau d'origine sont conservées, que 200 % d'interconnexions sont ajoutés, et que le nombre de processeurs supplémentaires est modulaire. On peut ainsi avoir un seul processeur supplémentaire comme une

ligne seulement ou une ligne et une colonne (non représenté figure 6),…. Toutefois, la taille de nos sous-réseaux est supposée assez faible et nous cherchons à limiter le surplus matériel. Nous nous limitons donc dans la suite à un maximum d'une ligne de processeurs supplémentaires.

2.3 Choix global

2.3.1 Introduction

En raison de la limitation du nombre d'entrées/sorties de chaque bloc aux entrées/sorties fonctionnelles, seul l'intérieur de chaque bloc peut être rendu reconfigurable de façon « fine ».

Toutefois, l'absence de reconfiguration globale peut entraîner une carence en fiabilité, à savoir une faute de la structure dès qu'un des blocs élémentaires est fautif ou qu'une interconnexion entre blocs élémentaires l'est. Pour palier cette difficulté, nous présentons une méthode de reconfiguration globale qui s'appuie sur une reconfiguration à gros grain. Elle servira également à optimiser l'utilisation du surplus matériel de la structure, notamment en cas d'amas de fautes très localisé. L'architecture ainsi que la méthode sont maintenant présentées.

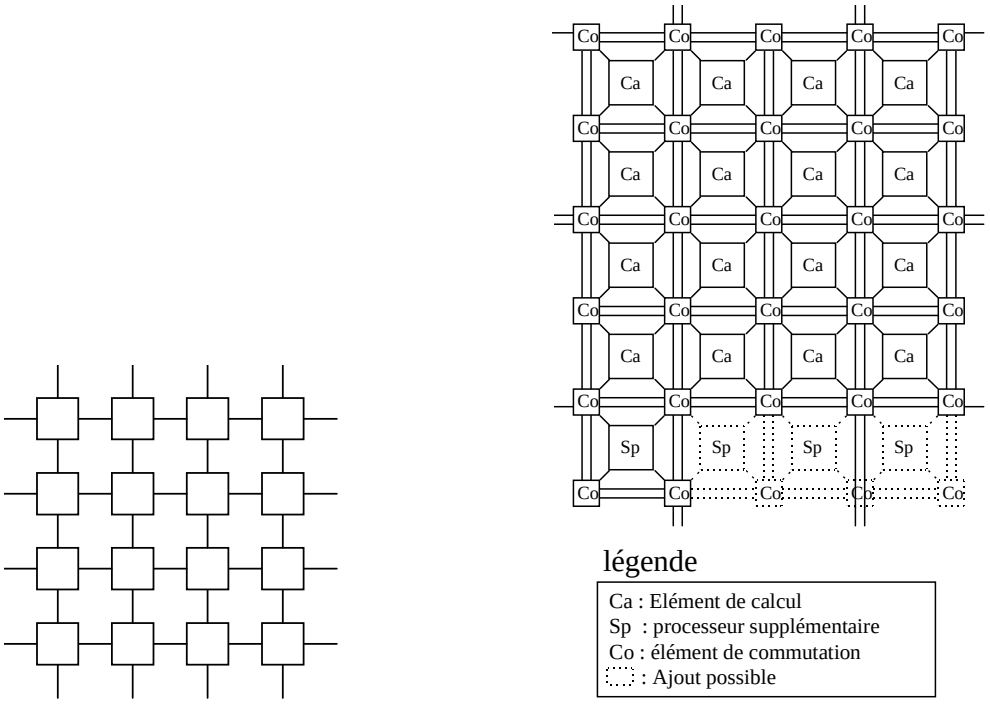


Figure 6 : exemple d'un sous-réseau 4*4 tolérant aux fautes (b) déduit d'un sous-réseau grille 2-D (a)

2.3.2 Architecture de reconfiguration globale

La reconfiguration globale repose sur une stratégie globale de remplacement d'une ligne ou d'une colonne de processeurs d'un bloc élémentaire par une ligne ou une colonne supplémentaire ou d'un

bloc élémentaire voisin. Le nombre total de lignes ou de colonnes supplémentaires que l'on s'accorde détermine alors le niveau de la tolérance aux fautes globale.

Nous proposons figure 7 l'architecture globale de reconfiguration. Dans cet exemple, chaque bloc élémentaire est constitué d'un sous réseau 4*4. Nous supposons que chacun des sous-réseaux permet la dérivation des processeurs en ligne (ce qui est le cas de la méthode de reconfiguration locale que nous proposons). Une colonne sur 8 du réseau global est considérée comme colonne supplémentaire. Les interconnexions associées à celle-ci sont également considérées comme supplémentaires.

Des éléments de commutation sont également ajoutés entre les lignes de blocs. Ces éléments permettent de commuter chaque colonne de processeurs d'un bloc élémentaire avec : soit la colonne correspondante du bloc élémentaire de la ligne inférieure ou supérieure, soit les colonnes à la gauche ou à la droite de cette colonne. Ils peuvent être réalisés à partir de simples commutateurs, et leur temps de traversée est alors très faible.

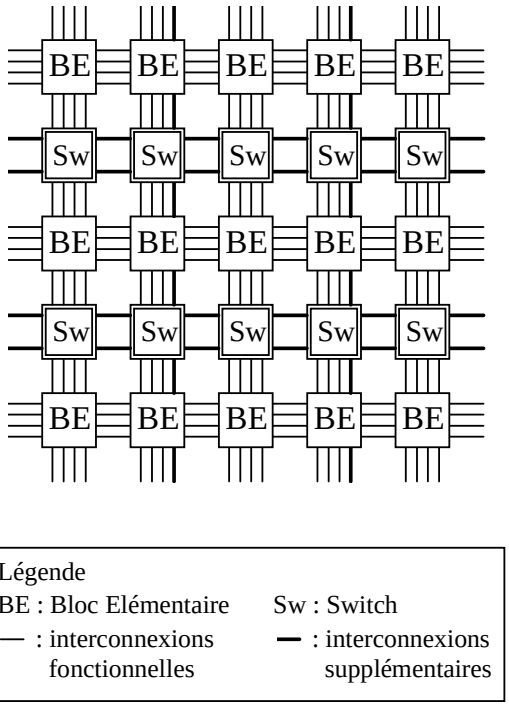


Figure 7 : principe de l'architecture de reconfiguration globale

Le fonctionnement de la tolérance aux fautes est le suivant : un certain nombre de colonnes supplémentaires étant définies pour tout le réseau, on peut utiliser une portion d'une de ces colonnes en cas de faute d'une interconnexion en colonne (les fautes dans les interconnexions des lignes ne sont pas tolérées dans ce cas) ou d'un bloc élémentaire : lorsqu'une colonne est logiquement supprimée d'un bloc élémentaire, ce bloc dispose alors d'une colonne physique de processeurs pouvant servir de processeurs supplémentaires pour la reconfiguration locale.

Lorsqu'une interconnexion est fautive, le choix de la colonne du bloc élémentaire à supprimer est évident. Lorsqu'un bloc élémentaire est fautif, nous choisissons de supprimer la colonne contenant le plus de processeurs fautifs. Une fois la colonne supprimée, on peut à nouveau tenter une reconfiguration locale du bloc concerné, avec une probabilité de réussite très importante. Plusieurs colonnes peuvent ainsi être supprimées logiquement d'un bloc dans lequel il y aurait un amas de fautes. Il ne faut toutefois pas oublier que la suppression de colonnes entraîne une augmentation de la longueur des chemins entre processeurs voisins. Ainsi, nous nous limiterons à la reconfiguration d'au plus 3 colonnes fautives jointes dans un bloc élémentaire. Cependant, un tel amas de fautes est peu probable, mis à part dans un cas catastrophique, dû, par exemple, à un fort court-circuit local.

La technique de reconfiguration globale que nous proposons s'apparente aux techniques de remplacement d'une ligne, mais possède plus de souplesse que celles-ci. En effet, lorsqu'une faute est détectée sur une colonne, nous ne dérivons pas la colonne entière du réseau, mais seulement une partie de celle-ci. Le grain de reconfiguration est donc plus petit dans notre cas, et les processeurs supplémentaires mieux utilisés.

Pour résumer, les objectifs de la reconfiguration globale que nous nous fixons sont de deux ordres :

- tolérance aux fautes des interconnexions entre blocs élémentaires ;
- tolérance aux fautes d'amas de fautes ou « points faibles » du réseau, cette dernière utilisation ne devant se faire qu'en cas d'échec de reconfiguration locale du bloc élémentaire.

2.3.3 Schémas de reconfiguration globale dérivés de l'architecture générale

Nous proposons deux schémas de reconfiguration globale. Le premier (schéma 1) consiste à considérer que chaque bloc élémentaire comporte une colonne ou ligne de processeurs supplémentaires servant à la reconfiguration locale ou globale (figure 8). Le surplus matériel en terme de processeurs est alors celui de la reconfiguration locale avec une colonne ou ligne de processeurs supplémentaires. Par contre, certaines interconnexions ne sont pas fonctionnelles. Ce sont donc des interconnexions supplémentaires. Dans le cas de la figure 8, nous avons 2 interconnexions supplémentaires pour 14 utiles, soit 14 %, ce qui est acceptable pour un bloc élémentaire de cette taille.

Lors d'une faute d'interconnexions verticales, il est alors possible de changer la colonne supplémentaire de place grâce à l'architecture de reconfiguration globale. Cette dernière est également utile si un bloc élémentaire particulier constitue un point faible important dans le système. Il est alors possible de tolérer jusqu'à 3 colonnes de processeurs fautifs comme il sera montré dans le paragraphe suivant, en utilisant les colonnes supplémentaires des blocs élémentaires à la gauche et à la droite du bloc incriminé.

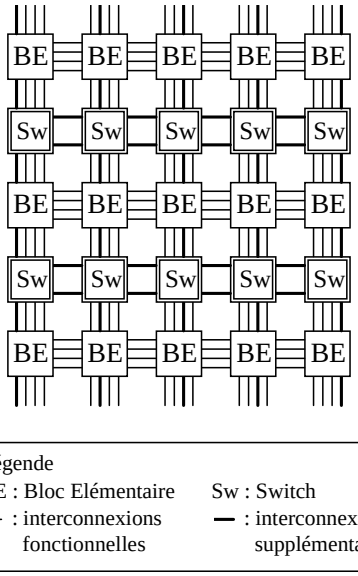


Figure 8 : schéma 1. Une colonne de processeurs de chaque bloc élémentaire est considérée comme colonne supplémentaire

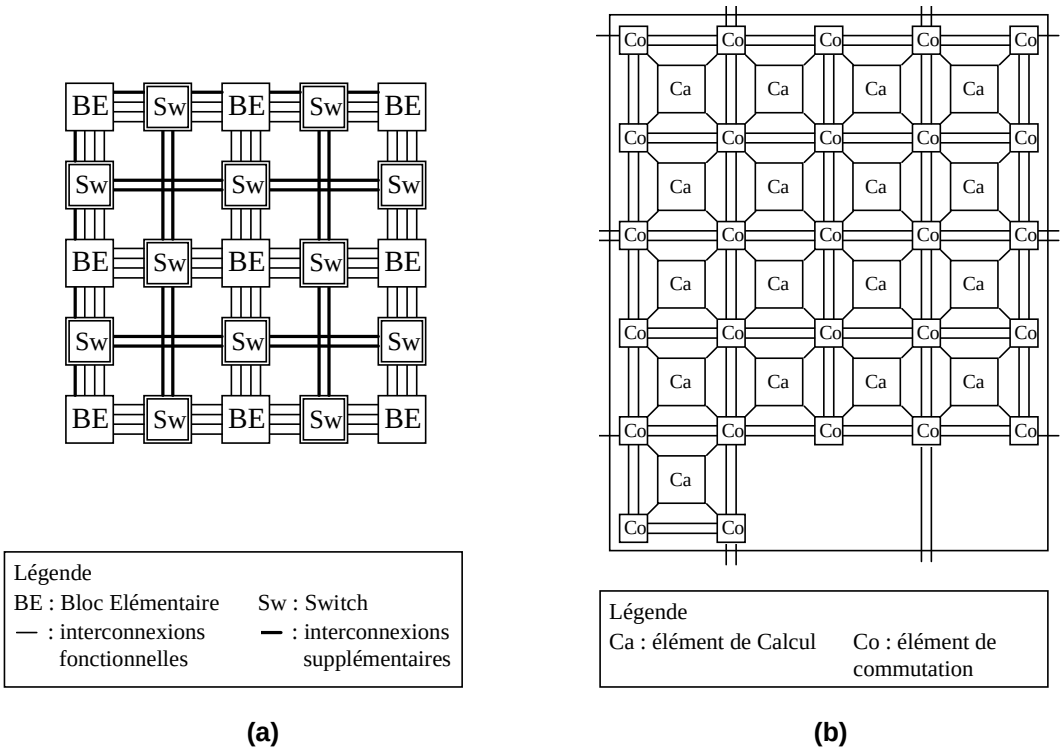


Figure 9 : schéma 2. (a) exemple d'un réseau de 3*3 blocs élémentaires comportant une ligne et une colonne de processeurs supplémentaire. (b) détail d'un bloc élémentaire comportant le réseau reconfigurable localement

Le deuxième schéma (schéma 2) utilise un nombre très restreint de lignes ou colonnes supplémentaires pour tout le réseau (voire une seule). Les reconfigurations locales et globales sont alors indépendantes. Ce schéma permet de limiter fortement le surplus matériel. Ainsi, une structure comportant, par exemple, un seul processeur supplémentaire par bloc élémentaire, une seule colonne supplémentaire ainsi qu'une seule ligne supplémentaire pour tout le réseau (voir figure 9) permet d'obtenir un double niveau de fiabilité pour un coût limité.

Il est à noter que le schéma 1 de reconfiguration ne sera utilisé que pour les structures à très forte demande en fiabilité en raison de son coût matériel élevé, alors que le schéma 2 sera utilisé pour les demandes moins fortes en fiabilité.

2.3.4 Technique de reconfiguration globale

2.3.4.1 Introduction

La technique de reconfiguration la plus simple (voir chapitre 4) consiste en une dérivation d'une ligne fautive vers une ligne supplémentaire. L'intérêt de cette solution est l'excellente conservation de la localité ainsi que la simplicité de la méthode et de l'architecture de reconfiguration. Par ailleurs, le surplus matériel nécessaire en termes d'interconnexions et de switches est très faible. Au niveau global, les deux premiers points sont extrêmement importants et rendent donc cette solution très attrayante. Toutefois, le point faible de cette méthode est son efficacité : le rapport taux de reconfiguration moyen sur coût matériel est très faible.

Dans notre cas, nous disposons de l'avantage du niveau de reconfiguration local. En effet, la méthode de reconfiguration que nous avons proposée pour l'intérieur des blocs élémentaires permet de supprimer une colonne logique du placement sans aucune difficulté. La dérivation des lignes est donc faite naturellement, et ne nécessite pas d'ajout matériel.

Pour la reconfiguration globale, il est donc possible de se limiter à un surplus matériel nul en dérivant une colonne entière du réseau. Cette solution est toutefois très gourmande en matériel pour une efficacité très médiocre puisqu'une colonne du réseau ne permet de tolérer qu'une seule faute d'interconnexion ou de bloc élémentaire. Nous allons montrer comment l'architecture de reconfiguration globale que nous proposons permet d'obtenir une meilleure utilisation moyenne des processeurs supplémentaires en diminuant la taille du grain de reconfiguration, et donc d'améliorer l'efficacité de la méthode.

2.3.4.2 Principe de reconfiguration : remplacements successifs et placement des colonnes

Le principe de remplacement d'une colonne « fautive » d'un bloc élémentaire par une colonne supplémentaire est très simple comme illustré figure 10a. La colonne est remplacée par sa voisine en direction de la colonne supplémentaire, et ainsi de suite jusqu'à atteindre la colonne supplémentaire comme pour les techniques de « fault-stealing » les plus simples [Lombardi1989].

La seule difficulté de la méthode concerne le choix de la direction de reconfiguration pour chaque colonne fautive : un mauvais choix peut entraîner une impossibilité de reconfiguration pour cette colonne ou une colonne fautive des blocs élémentaires des lignes supérieures ou inférieures (figure

10b). En effet, la simplicité des switches impose que toutes les colonnes des lignes supérieure et inférieure d'une colonne fautive soient remplacées dans la même direction que celle-ci. Il est toutefois possible de passer outre cette contrainte, en « redressant » la structure (figure 10c), c'est à dire en modifiant l'emplacement d'une partie de la colonne supplémentaire du réseau. La complexité de l'algorithme nécessaire afin d'effectuer ce « redressement » lorsque cela est nécessaire est importante. La simplicité du schéma de reconfiguration au niveau global étant nécessaire, l'algorithme de « redressement » est écarté. Nous utiliserons toutefois une version modifiée du simple algorithme de remplacement successif en incluant un choix, pour chaque colonne fautive, de la direction de reconfiguration.

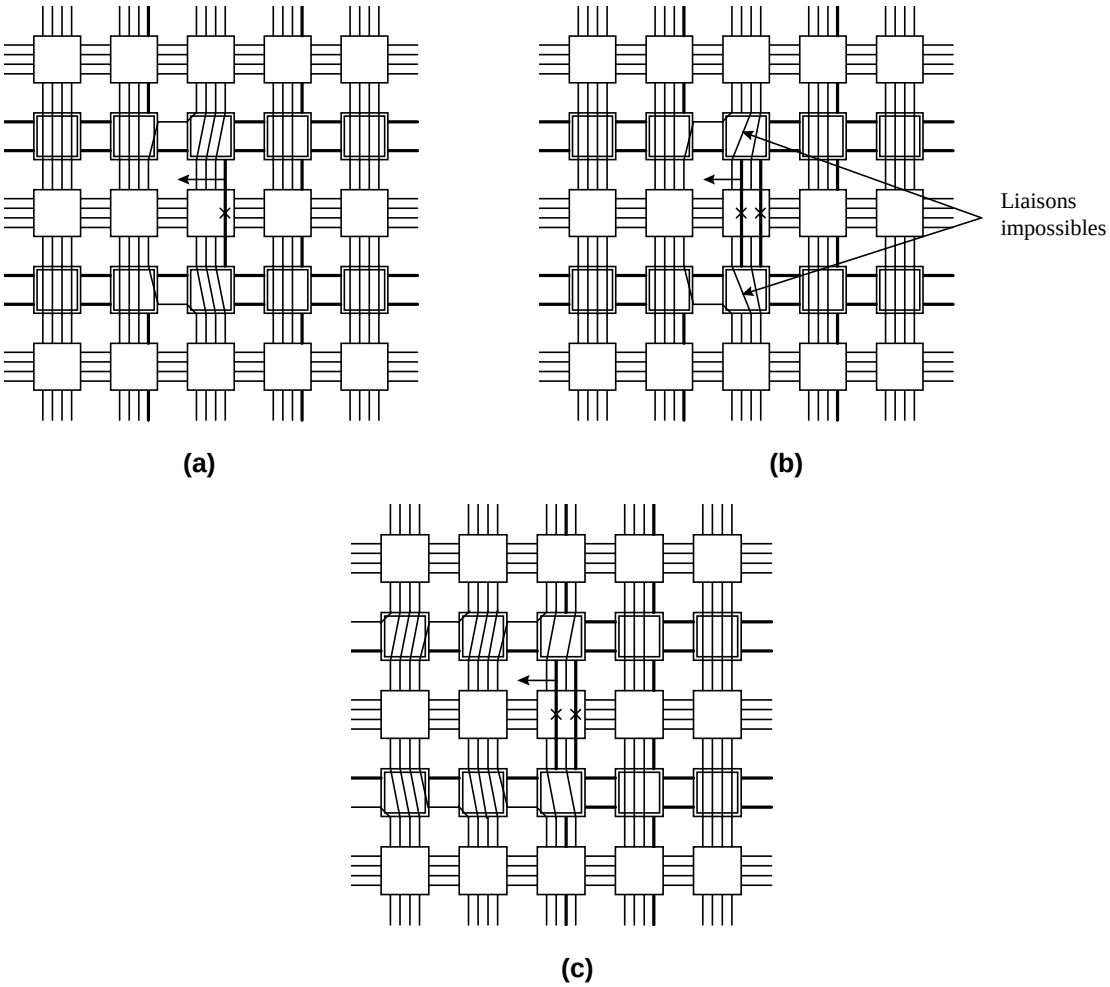


Figure 10 : illustration du principe de reconfiguration. (a) une colonne fautive d'un bloc élémentaire est remplacée par sa voisine jusqu'à une colonne supplémentaire (la flèche indique la direction de reconfiguration). (b) exemple d'échec de reconfiguration dû à un mauvais choix de la direction de reconfiguration. (c) solution de « redressement » permettant d'éviter le problème de recouvrement des directions de reconfiguration

Le problème concernant la direction de reconfiguration implique celui du placement des colonnes supplémentaires. Lorsque deux colonnes supplémentaires ou plus sont utilisées, il est nécessaire d'assurer que chaque colonne a deux directions de reconfiguration possibles. Ainsi, deux colonnes doivent être placées aux deux frontières du réseau pour une grille. En l'absence d'hypothèse de répartition des fautes, les autres colonnes supplémentaires (s'il y en a) sont réparties régulièrement entre ces deux premières colonnes.

2.3.4.3 Algorithmes de reconfiguration

Un premier algorithme A_1 très simple peut être déduit d'une structure ne disposant que d'une seule colonne supplémentaire dans le réseau. Celle-ci est placée à l'une des extrémités et le remplacement s'effectue toujours dans la même direction. A chaque colonne est associée un bit indiquant si elle est commutée ou non. Lorsqu'une colonne fautive est détectée par l'unité de contrôle, celle-ci envoie un signal de commutation à toutes les colonnes dans la direction de reconfiguration, ceci jusqu'à la colonne supplémentaire. La reconfiguration ne devient impossible que lorsque la commutation d'une colonne d'un bloc élémentaire a déjà été effectuée, c'est à dire qu'il y a plus d'une colonne fautive pour une ligne de blocs élémentaires. Par conséquent, cet algorithme est optimal dans le sens où il permet le remplacement d'une colonne quelconque de chaque ligne.

Un algorithme A_2 plus complexe doit être mis en place lorsque deux colonnes supplémentaires ou plus sont mises en œuvre. Afin de simplifier l'explication (ce qui n'enlève rien à sa généralité), nous prenons le cas de deux colonnes supplémentaires, qui sont placées de chaque coté du réseau. A chaque colonne d'un bloc élémentaire est associé un nombre compris entre 0 et 2 indiquant l'état de commutation *ec*. 0 indique une commutation gauche, 1 une commutation avec le vis-à-vis direct et 2 une commutation droite. Le réseau et l'algorithme sont initialisés en associant la valeur 1 à *ec* pour chaque colonne de blocs élémentaires. D'autre part, on utilise un bit indiquant la direction de commutation *dc* (gauche ou droite). On initialise *dc*, soit à la même valeur pour toutes les colonnes, soit à la valeur correspondant à la direction de la plus proche colonne supplémentaire.

La reconfiguration peut alors être réalisée localement (par un module matériel ou une routine de programme des processeurs) à partir de la colonne fautive ou globalement (à partir de l'unité centrale), en scrutant tout le réseau à chaque apparition d'une nouvelle colonne fautive, cette deuxième solution donnant de meilleurs résultats.

Le fonctionnement de l'algorithme A_2 réalisé localement est très simple : on tente de reconfigurer le réseau en procédant comme pour l'algorithme A_1, dans la direction *dc,* et en modifiant *ec* en conséquence. Si la commutation est impossible (*ec* prend une valeur interdite), on change alors la direction *dc*. Si les deux directions entraînent une impossibilité de reconfiguration, l'algorithme échoue. Cet algorithme très simple et très rapide donne malheureusement un assez mauvais taux de reconfiguration, car les directions de reconfigurations ne sont pas optimisées.

L'algorithme A_2 réalisé de manière globale vérifie d'abord que la commutation est possible avant de lancer une reconfiguration en indiquant à chaque colonne fautive sa direction de remplacement. Il est basé sur une scrutation du réseau colonne par colonne, de la gauche vers la droite. Chaque colonne

possède au départ la direction gauche de reconfiguration. Pour chaque colonne fautive, on détermine alors la direction de remplacement :

Pour chaque colonne de blocs élémentaires fautive C_i

Si la direction de C_i est gauche :

S'il n'y a pas de colonne fautive à gauche

passer à la colonne suivante ;

Sinon

direction = droite ;

S'il n'y a pas de colonne fautive à droite de C_i :

Pour les colonnes des blocs situées sur les lignes supérieures et inférieures de la ligne courante et à droite de C_i :

direction = droite ;

passer à la colonne suivante ;

Sinon

reconfiguration impossible : plus de deux colonnes fautives ;

Si la direction de C_i est droite :

S'il n'y a pas de colonne fautive à droite de C_i :

Pour les colonnes des blocs situées sur les lignes supérieures et inférieures de la ligne courante et à droite de C_i :

direction = droite ;

passer à la colonne suivante ;

Sinon :

reconfiguration impossible : incompatibilité de direction entre deux lignes.

L'algorithme, dans ce cas, est optimal pour deux colonnes, par rapport aux moyens de reconfiguration. La reconfiguration peut toutefois échouer dès lors que 4 colonnes sont fautives (deux colonnes situées dans deux lignes adjacentes de blocs élémentaires).

Les algorithmes de reconfiguration pour plus de deux colonnes supplémentaires sont rigoureusement les mêmes que ce dernier. Toutefois, si on utilise pour 3 colonnes supplémentaires le même algorithme que pour 2 colonnes supplémentaires, cet algorithme ne sera pas optimal par rapport aux moyens de reconfiguration.

Enfin, les mêmes algorithmes peuvent être utilisés pour une reconfiguration à la fois en ligne et en colonne.

2.3.5 Conclusion

Nous avons proposé dans cette section une architecture et une méthode de reconfiguration globale à gros grain. L'approche est inspirée des méthodes de dérivation d'une ligne ou colonne d'un réseau. Elle se différencie toutefois nettement de ces méthodes par une réduction de la taille du grain de

reconfiguration : elle est en effet limitée à une colonne d'un bloc élémentaire (au lieu d'une colonne du réseau entier). Par conséquent, la reconfiguration est de meilleure qualité. Il faut toutefois noter que nous avons pu améliorer cette reconfiguration grâce à l'utilisation du niveau local de reconfiguration. Ainsi, dans notre approche, ces deux niveaux sont fortement liés.

2.4 Conclusion

Nous avons proposé une architecture complète de reconfiguration. En raison du bloc élémentaire (ASIC, MCM ou carte) dont la bande passante supplémentaire disponible est très faible, nous avons envisagé une technique de reconfiguration à deux niveaux : une reconfiguration à grain fin modulaire en nombre de processeurs supplémentaires pour le réseau interne au bloc élémentaire (niveau local), et une reconfiguration à gros grain basée sur le remplacement d'une colonne entière d'un bloc élémentaire (niveau global).

La reconfiguration au niveau local est transparente vis-à-vis de la structure complète. En effet, *aucune entrée ou sortie* n'est ajoutée au bloc élémentaire. Ainsi, le schéma logique du bloc élémentaire, vu de l'extérieur, reste rigoureusement le même. La reconfiguration est réalisée soit par les processeurs valides du bloc élémentaire, soit par un bloc spécifique lorsque le nombre de processeurs supplémentaire est faible, soit par un ordinateur « hôte » qui peut être, par exemple, le processeur séquentiel maître d'un sous-réseau SIMD dans le cas d'une structure Multi-SIMD.

Le niveau de reconfiguration globale se doit d'être beaucoup plus simple, et de minimiser les pertes en performance engendrées par une reconfiguration entre deux blocs élémentaires, le chemin de donnée entre deux de ces blocs étant critique. Nous proposons donc une architecture de reconfiguration globale très simple qui peut être réalisée soit de façon locale, au niveau de chaque élément de commutation, soit de façon globale, dirigée par un ordinateur hôte.

Le niveau de reconfiguration locale permet alors de tolérer les fautes des processeurs et des interconnexions internes au bloc élémentaire, alors que la reconfiguration globale tolère les fautes d'interconnexion entre les blocs élémentaires et pallie les difficultés liées à l'apparition d'amas de fautes dans le réseau.

D'autres méthodes que celle que nous proposons associent une reconfiguration locale et globale. Toutefois, et contrairement aux autres méthodes, les reconfigurations locale et globale sont étroitement couplées dans notre approche. Ainsi, la combinaison de ces deux niveaux donne une architecture particulièrement fiable : la reconfiguration globale peut « alimenter » en processeurs supplémentaires les blocs défaillants, pour que la reconfiguration locale répartisse correctement ceux-ci.

La section suivante se base sur le niveau local de reconfiguration et en déduit différentes architectures.

3 Architectures dérivées

3.1 Introduction

L'architecture locale de tolérance aux fautes utilise 200 % d'interconnexions supplémentaires. Ce surplus peut s'avérer trop important, et on peut désirer le réduire. Ceci est particulièrement vrai lorsque les interconnexions sont réalisées par des bus mono-directionnels (et donc doublées par rapport aux communications bi-directionnelles).

Nous proposons, dans cette section, deux architectures dérivées de l'architecture interne au bloc élémentaire. La première prend en compte les liaisons mono-directionnelles et montre qu'avec la méthode de reconfiguration que nous proposons, il est possible de réduire le surplus d'interconnexions de 50 %. La seconde architecture suppose que les processeurs de la structure ne peuvent pas communiquer en même temps dans les 4 directions nord-est-sud-ouest : nous proposons alors une architecture à communications réduites dynamiquement configurable.

3.2 Cas d'interconnexions mono-directionnelles

L'architecture du bloc élémentaire comporte 200 % d'interconnexions supplémentaires. Ce surplus d'interconnexions est tout de même assez important, et il semble difficile de le minimiser sans diminuer fortement les taux de reconfiguration.

Un cas particulier de l'architecture est obtenu lorsqu'on considère des liaisons mono-directionnelles. La figure 12 représente un exemple d'architecture tolérante aux fautes à liens mono-directionnelles possédant une ligne de processeurs supplémentaires. La méthode de reconfiguration locale autorise alors à supprimer certaines des interconnexions supplémentaires.

Le premier type d'architecture déduit est illustré par la figure 13. Le fait d'ajouter une ligne de processeurs supplémentaires implique un routage orienté plutôt haut/bas pour la structure. L'architecture exploite ce fait en conservant tous les liens orientés haut/bas et en supprimant la moitié des liens gauche/droite. On supprime ainsi 50 % d'interconnexions supplémentaires. Ce premier type d'architecture donne de très bons résultats de reconfiguration comme il le sera montré dans la suite.

Le second type d'architecture est basé sur une propriété de l'algorithme de routage que nous utilisons : celui-ci essaye de façon systématique tous les chemins possibles entre processeurs voisins. On peut donc supposer que si un routage est impossible par manque de liens à un endroit, ce routage pourra se faire à un autre endroit plus dense en liens supplémentaires. L'architecture de la figure 14a illustre cette idée par un partage diagonal des liens alors que celle de la figure 14b utilise un partage régulier des liens entre les switches. Nous verrons que ces deux architectures donnent également de bons taux de reconfiguration.

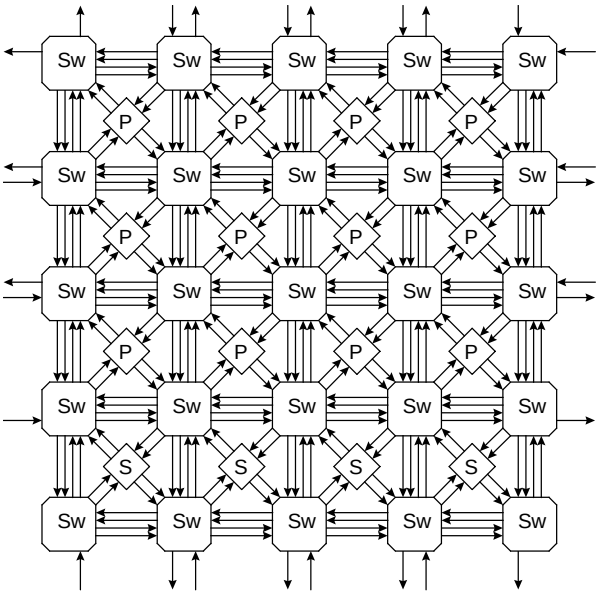


Figure 12 : réseau comprenant 200 % d'interconnexions supplémentaires par rapport à la grille 2-D. Les processeurs (P) possèdent 4 entrées et 4 sorties diamétralement opposées qui sont reliées au switches (Sw). Le placement de ces entrées et sorties simplifie le routage des switches. Une seule ligne de processeurs supplémentaires (S) est ajoutée

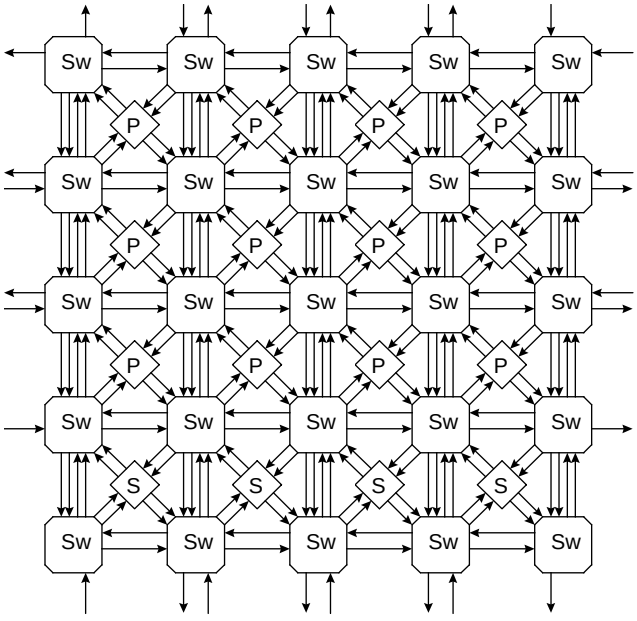
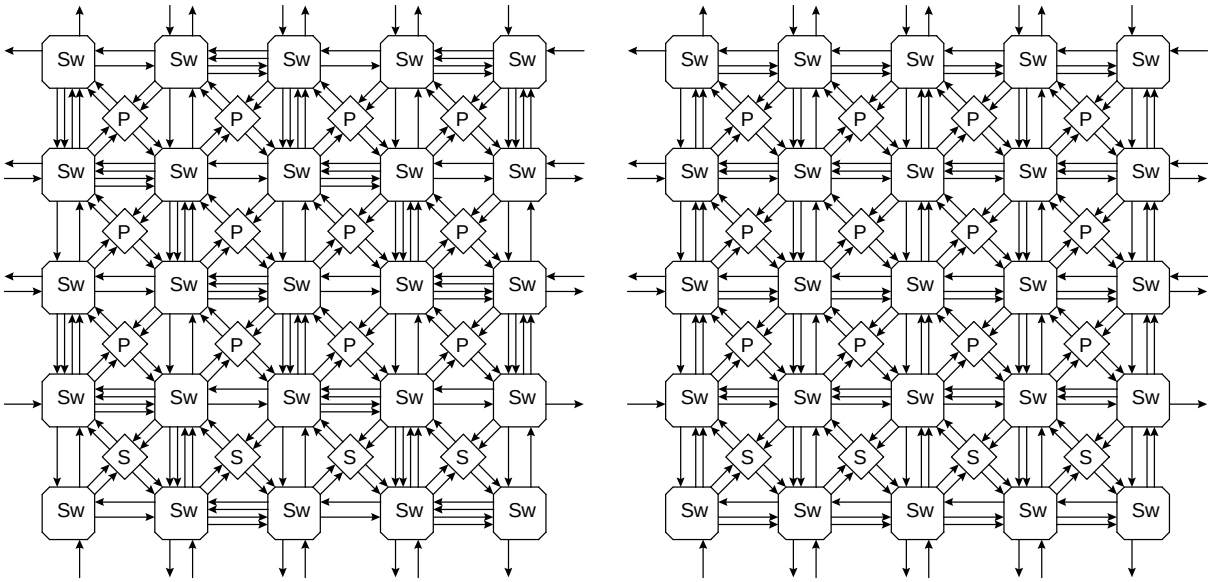


Figure 13 : premier type d'architecture déduite de la figure 12 : la moitié des liens gauche-droite est supprimée



a) réseau en « liens diagonaux » **b)** réseau en « liens réguliers »

Figure 14 : deuxième catégorie d'architectures. Une partie des liens est conservée par endroit de façon à favoriser le routage des switches

Ces trois réseaux possèdent des caractéristiques différentes en vue de leur implémentation. Le réseau de la figure 13 possède des éléments d'interconnexion identiques, ce qui lui procure un avantage certain sur les deux autres architectures. Le réseau de la figure 14b est le seul à incorporer une régularité du nombre d'interconnexions dans les deux dimensions. Au contraire, l'irrégularité du réseau de la figure 14a tant au niveau interconnexions qu'au niveau des switches en fait le plus mauvais candidat.

3.3 Réseau réduit

Selon les processeurs utilisés, les communications simultanées des 4 entrées et 4 sorties (communications 4-ports) peuvent ne pas être nécessaires ou possibles. Ainsi, nous proposons un réseau dont les possibilités de communications sont réduites par rapport au réseau original de grille 2-D. Ce réseau présente l'avantage d'avoir un très faible surcoût en interconnexions par rapport au réseau grille 2-D (de l'ordre de 50%) tout en offrant des possibilités de communication intéressantes.

Le réseau réduit est présenté figure 15. Comme on peut le constater, les entrées et sorties des processeurs sont limitées à deux contre 4 pour la grille traditionnelle. Afin de conserver toutes les possibilités logiques de communication de la grille, le réseau a été rendu dynamiquement reconfigurable. Les changements de configuration s'effectuent au niveau de la programmation des switches. Ainsi, le switch n'est plus uniquement lié à la reconfiguration, mais devient un élément à part entière du réseau fonctionnel des communications. Nous réalisons ainsi un premier pas vers l'unification de deux concepts de reconfiguration, le premier ayant pour but de modifier la fonction du réseau, et le second axé sur la tolérance aux fautes.

Afin d'obtenir la souplesse nécessaire, des registres de configuration de 18 bits ont été ajoutés aux switches. Les configurations fonctionnelles sont alors chargées dans ces registres à l'initialisation du système, puis sont modifiées à chaque reconfiguration lorsqu'un processeur fautif a été découvert. Lors du fonctionnement normal du système, une partie des instructions parvenant aux processeurs est destinée au choix de la communication et sélectionne le registre de configuration adéquat des switches. Les configurations fonctionnelles sont donc dynamiquement sélectionnées en un seul cycle.

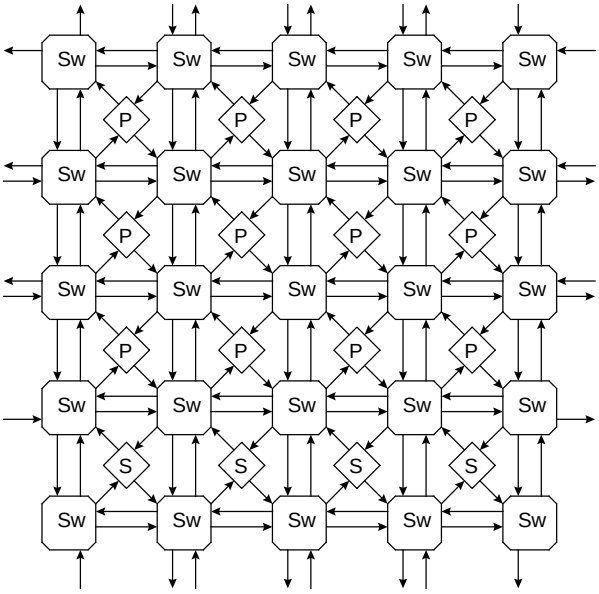


Figure 15 : réseau réduit

Les différentes possibilités de communication du réseau réduit sont montrées figure 16. Dans les deux premières configurations (figure 16a) le processeur peut recevoir deux données par cycle tout en en émettant une. Dans ce cas, en deux cycles au maximum, l'élément est sûr de recevoir la donnée d'un de ses voisins, quelle que soit sa provenance, et d'émettre sa donnée disponible vers le voisin demandé. Si les communications sont régulières, c'est à dire que chaque élément de calcul reçoit, en même temps, la même donnée (cas de certains calculs de matrice, de certains traitements d'image de bas niveau, de corrélations...), alors, ces communications sont suffisantes et on conserve la complète fonctionnalité du système. Dans le cas contraire, il est possible d'ajouter de nouvelles communications fonctionnelles pour diminuer la pénalité. Deux possibilités intéressantes de configurations sont présentées figure 16b et 16c et un partitionnement du réseau en anneau est proposé figure 17.

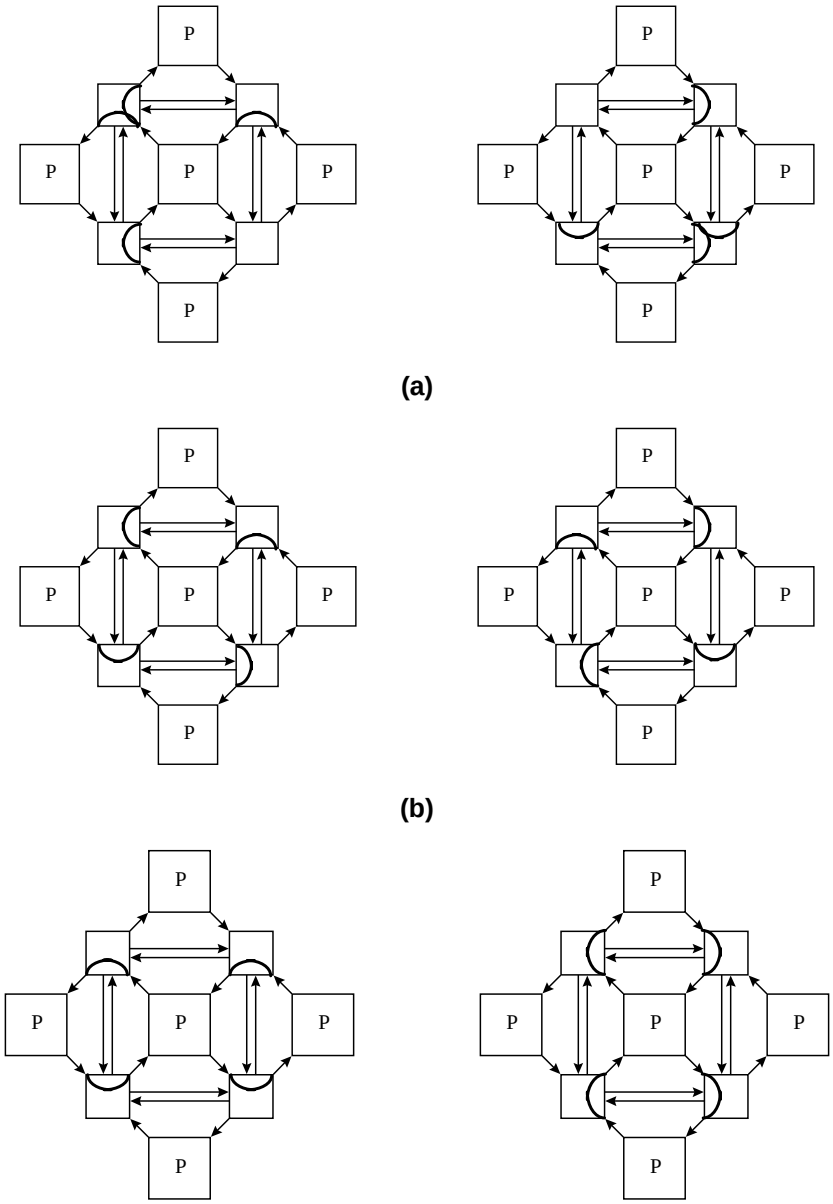


Figure 16 : configurations possibles du réseau réduit. (a) cas deux entrées/une sortie par élément de calcul. (b) communications croisées et (c) communications en ligne ou en colonne

Dans le premier cas (figure 16b), chaque élément de calcul reçoit des communications de ces voisins droit et gauche (respectivement haut et bas) et émet ses résultats en direction de ces voisins haut et bas (respectivement droit et gauche) en un cycle. Ce fonctionnement permet, par exemple, de disposer pour chaque élément de calcul de données de ces 8 plus proches voisins en un nombre minimal de cycles (4).

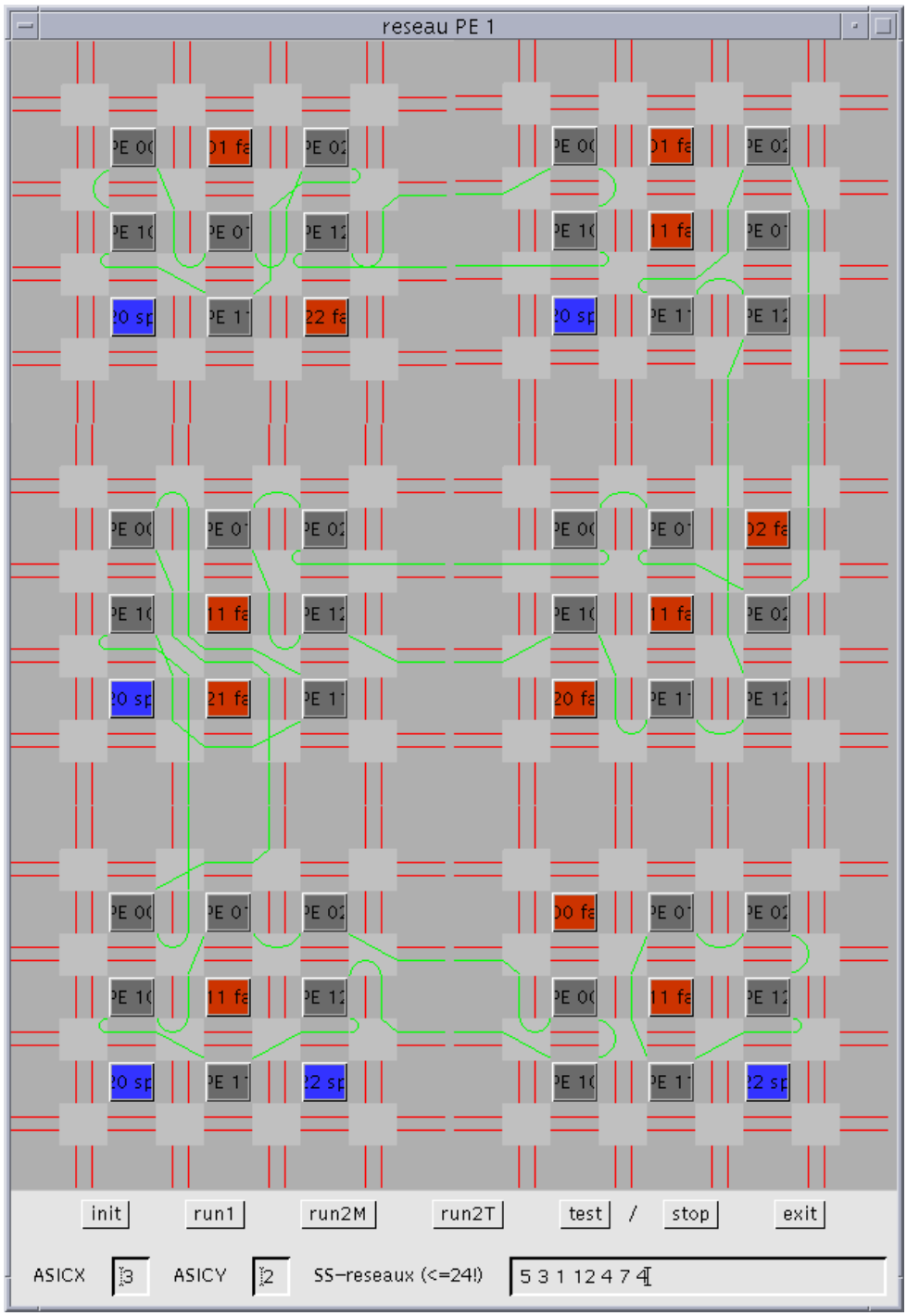


Figure 17 : prototype de fonctionnement en anneaux indépendants du réseau réduit

Dans le deuxième cas (figure 16c), chaque élément de calcul peut émettre et recevoir des données de ses voisins gauche et droite (resp. haut et bas). Nous obtenons ainsi des communications bidirectionnelles en ligne (resp. en colonne). Il est à noter que cette configuration peut être utilisée dans le réseau général. Elle mène alors à un doublement du débit des communications en lignes ou en colonnes, ce qui peut permettre d'améliorer la vitesse d'algorithmes utilisant beaucoup ce type de communications.

Enfin, il est possible, en associant les blocs élémentaires, de grouper les processeurs en anneaux de différentes tailles (figure 17). La souplesse de programmation des switches permet alors de faire des groupements de processeurs pairs ou impairs tolérants aux fautes parmi les processeurs de chaque anneau.

3.4 Conclusion

La méthode de reconfiguration locale proposée au chapitre 5 permet de modifier le réseau d'interconnexions des architectures. Dans cette section, deux types de réseaux utilisant cet avantage sont présentés. Le premier réduit les interconnexions supplémentaires d'un réseau grille à liens monodirectionnels. Le second réduit de façon encore plus importante les interconnexions et la complexité des éléments de commutation pour aboutir à un réseau réduit, fonctionnellement reconfigurable, dédié principalement aux applications à communications régulières. Ce dernier type de réseau ouvre la voie vers un champ d'investigation très large qui consiste à réunir en une seule approche la reconfiguration dynamique de réseaux fonctionnellement reconfigurables et la reconfiguration statique d'un réseau structurellement tolérant aux fautes.

4 Quelques schémas de tolérance aux fautes

4.1 Introduction

Les techniques de reconfiguration et les architectures correspondantes présentées ci-dessus sont modulaires et adaptables aux différents besoins en fiabilité désirés. Il est ainsi possible d'imaginer plusieurs schémas de tolérance aux fautes, à partir d'un schéma minimal jusqu'à un schéma plus complexe et plus complet. Nous en présentons maintenant quelques exemples.

4.2 Schémas minimums

Il est possible que, pour des raisons de surplus matériel, on soit amené à prendre une configuration minimale de tolérance aux fautes. Deux schémas minimums sont alors possibles, ils peuvent se résumer ainsi :

1. Reconfiguration locale seule. Un seul processeur par bloc élémentaire est reconfigurable (on a un seul processeur supplémentaire). Ce cas suppose que les fautes possibles sont uniformément réparties. Les fautes d'interconnexions entre blocs ne sont pas prises en compte.

2. Reconfiguration globale comportant une colonne et une ligne de processeurs supplémentaires et reconfiguration locale sans processeur supplémentaire. Ce schéma permet de tolérer des fautes d'interconnexions verticales et horizontales entre blocs élémentaires, ainsi qu'une faute par ligne et une faute par colonne du réseau. Ce schéma suppose que la structure est assez sûre et qu'on désire uniquement se prémunir contre des fautes rares mais pouvant survenir dans n'importe quelle partie de la structure.

4.3 Schémas intermédiaires

Ces schémas permettent une meilleure prise en compte de fautes multiples pouvant affecter le réseau global.

1. Reconfiguration locale seule avec de 2 processeurs à une ligne ou colonne de processeurs supplémentaires par bloc élémentaire. C'est le schéma prenant le mieux en compte le rapport surplus en surface / objectif de fiabilité. Ce cas sera repris dans les calculs de fiabilité par la suite.

2. Reconfiguration globale avec une ligne et/ou colonne de processeurs supplémentaires et reconfiguration locale avec un seul processeur supplémentaire par bloc élémentaire. Ce cas permet de prendre en compte des fautes se produisant peu souvent, mais susceptibles de se produire dans plusieurs blocs élémentaires de façon uniforme. Il permet également de reconfigurer un amas local de fautes (plusieurs processeurs fautifs dans un seul bloc élémentaire) et quelques fautes d'interconnexion entre blocs élémentaires.

4.4 Schéma complet

Lorsque le besoin en fiabilité est très important et qu'aucune partie du réseau n'est considérée comme fiable, on peut utiliser le schéma suivant, assez coûteux en matériel supplémentaire :

1. Reconfiguration locale avec une ligne ou colonne de processeurs supplémentaires et reconfiguration globale comportant au moins une ligne et une colonne de processeurs supplémentaires. Dans ce cas, un nombre important de processeurs peut être fautif par bloc élémentaire et on tolère des fautes d'interconnexion horizontales et verticales entre blocs élémentaires ainsi que des amas de fautes du circuit.

Ce schéma le plus complet est également le plus complexe et le plus lourd matériellement à mettre en œuvre. Il convient toutefois à des systèmes à forte demande en fiabilité.

4.5 Conclusion

Nous avons proposé, à titre d'exemples, quelques schémas de reconfiguration. Ceux-ci correspondent à des demandes en fiabilité très différentes qui varient selon la réalisation du circuit et les besoins de l'application. Le tableau 2 résume les caractéristiques de chaque schéma. Il montre que les techniques de reconfiguration proposées sont complètes puisqu'elles permettent de répondre à la plupart des demandes en fiabilité de systèmes non critiques.

Schéma	complexité	surplus matériel	utilisation du surplus	tolérance aux fautes (* peu adapté, **** très adapté)		
	* ⇨ **** très faible ⇨ très forte	* ⇨ **** faible ⇨ important	* ⇨ **** mauvaise ⇨ très bonne	interconnexions	Pr : fautes uniformes	Pr : fautes groupées
min. 1	**	* à **	**	NON	***	NON
min. 2	*	**	*	1/ligne et/ou 1/colonne	*	*
interm. 1	** à ***	** à ***	***	NON	****	***
interm. 2	**	** à ***	***	1/ligne	***	****
comp.	***	****	***	> 1/ligne et 1/colonne	****	****

Tableau 2 : caractéristiques de quelques schémas de reconfiguration possibles

5 Résultats

5.1 Introduction

Les architectures de tolérance aux fautes globales et locales ainsi que les algorithmes de reconfiguration associés ont été définis dans les parties précédentes. Il est maintenant important de mesurer l'impact de ces techniques tant au niveau du coût matériel et des performances que de l'augmentation en fiabilité obtenue. Certains résultats ont déjà été abordés tout au long des sections précédentes. Ils sont maintenant repris ici afin de les synthétiser.

5.2 Surplus matériel et pertes en performance

5.2.1 Introduction

Le surplus matériel provient de trois éléments :

- les processeurs supplémentaires ajoutés, (surplus le plus important) ;
- les éléments d'interconnexion ou switches ;
- les interconnexions supplémentaires.

De manière générale, le surplus matériel engendré a une influence directe sur les performances de la structure. En effet, une augmentation de la taille du circuit entraîne des chemins plus longs et par conséquent une perte en performance. Cette dernière est difficilement quantifiable, car elle est complètement dépendante de l'architecture des processeurs ainsi que des méthodes de communication. Il est par contre beaucoup plus facile de quantifier le temps de traversée des switches utilisés, qui constituent une cause importante de la baisse de performance due à la tolérance aux fautes.

Chacun des trois éléments constituant le surplus matériel sont maintenant repris, avec une analyse des temps de traversée pour les éléments d'interconnexion.

5.2.2 Processeurs supplémentaires

Le tableau 3 résume le surplus matériel pour les 5 schémas de tolérance aux fautes proposés dans la section précédente. Le surplus est indiqué en terme de lignes L et de colonnes C de processeurs, ainsi que de nombre B de blocs élémentaires. Afin de fournir une indication plus précise du surplus matériel que chacun des schémas représente par rapport aux autres, 2 exemples de structures sont donnés. Dans le premier cas, nous supposons que le système est constitué de 64 ASIC répartis sur 8 lignes et 8 colonnes et comportant chacun un réseau de 4*4 processeurs, formant un calculateur parallèle de 1024 processeurs. Dans le second cas, nous supposons que le système est composé de 16 ASIC de 8*8 processeurs, soit également 1024 processeurs. Nous supposons également que le schéma intermédiaire 1 comporte 2 processeurs supplémentaires par ASIC.

Schéma	Surplus matériel	Cas 1 : 64 * (4*4)			Cas 2 : 16 * (8*8)		
		PU	PS	PS/PU	PU	PS	PS/PU
min. 1	B	1024	64	6,25 %	1024	16	1,56 %
min. 2	L+C	960	64	6,67 %	960	64	6,67 %
interm. 1	2*B	1024	128	12,5 %	1024	64	3,12 %
interm. 2	L+C+B	960	128	13,3 %	960	80	8,33 %
comp.	C*B+L+C	992	320	33,3 %	992	192	19,35 %

Tableau 3 : surplus matériel des différents schémas. PU = nombre de processeurs utiles, PS = nombre de processeurs supplémentaires

Le surplus matériel de la méthode de reconfiguration globale ne varie pas selon la structure (schéma min. 1) alors que celui de la reconfiguration globale décroît quand le nombre de processeurs de l'ASIC croît. Nous verrons plus tard que la fiabilité de la structure dépend également du nombre de processeurs de l'ASIC.

5.2.3 Eléments d'interconnexion

Les switches utilisés dans les deux schémas de reconfiguration locale et globale doivent répondre à des critères très différents en terme de possibilités de communication. Le temps de traversée des switches de la reconfiguration globale est très faible alors que celui, beaucoup plus complexe, de la reconfiguration locale est à mesurer avec la plus grande attention. Nous nous poserons également la question du temps de la commutation des configurations de l'architecture réduite dynamiquement reconfigurable.

Les switches utilisés pour la méthode de reconfiguration locale possèdent plus ou moins d'entrées/sorties selon leur place dans l'architecture (intérieur du réseau ou frontière du réseau). Nous nous référerons uniquement au switch le plus complet, le plus défavorable.

Pour le réseau de grille 2-D avec liens mono-directionnels, le réseau le plus lourd en interconnexions que nous proposons, les switches sont composés de 10 bus en entrée et 10 bus en sortie. Nous avons donc évalué de tels switches en les implémentant avec 10 multiplexeurs 10 vers 1, chacun étant commandé par 4 bits de configuration. La synthèse de cette entité avec des bus de 32 bits a été faite avec l'outil de Synopsis, en technologie HCMOS7 de largeur de grille de 0,7µm. Cette synthèse a donné les résultats suivants : le switch est composé de 9510 portes équivalentes, soit 38000 transistors, et le temps de traversée est évalué à 1,4 ns. Afin d'affiner ces résultats, le switch a ensuite été placé et routé, puis les résultats obtenus ont servi à rétro-annoter le schéma de synthèse au niveau des résistances et des capacités. Avec 6 niveaux de métaux, le temps de traversée du switch est alors de 1,65 ns pour une surface couverte de 0,73 mm^2.

Le switch du réseau réduit comprend, dans sa forme la plus complète, 6 entrées et 6 sorties de 32 bits. Ce switch est alors composé de 6 multiplexeurs 6 vers 1, chacun étant contrôlé par 3 bits de commande. Nous en avons alors effectué la synthèse en reprenant les mêmes contraintes que celles de la synthèse précédente. Le switch obtenu est nettement moins important que le précédent, puisqu'il comprend 3634 portes équivalentes, soit 14540 transistors, pour une surface de 0.28 mm^2. Son temps de traversée est alors estimé à 1,17 ns.

Ces essais de synthèse montrent que la surface d'un switch croît de façon très importante lorsqu'on augmente le nombre d'entrées/sorties, puisqu'un switch à 10 entrées est environ 160 % plus gros qu'un switch à 6 entrées alors qu'il ne comprend que 67 % d'entrées supplémentaires. Le temps de traversée, par contre, n'augmente pas considérablement : celui d'un switch 10/10 n'est que 20 % plus important que celui d'un switch 6/6. Ces chiffres confirment les choix qui ont été faits pour l'architecture, puisque des switches plus complexes détériorent plus faiblement les temps de transit des données qu'ils n'améliorent les possibilités de communication : il est donc préférable de traverser quelques switches complexes plutôt qu'un grand nombre de switches plus simples.

Afin d'évaluer plus précisément les pertes en performance, une petite digression sur les temps de transit des données est maintenant nécessaire. Le temps de traversée d'un switch en technologie 0,7µm est d'environ 1,5 ns. Lorsque 4 switches sont traversés par les liens de communication de deux processeurs voisins, le temps de traversée des switches est donc de 6 ns. Nous pouvons comparer ce temps à celui nécessaire pour entrer et sortir des différents niveaux hiérarchiques du système (ASIC, MCM ou PCB) : le temps maximal nécessaire pour aller d'une carte à l'autre est estimé à 40 ns [Scheer1997]. Il est donc nettement supérieur au temps de traversée des 4 switches. Les switches ont par ailleurs été conçus dans une « vieille » technologie, et les temps de traversée auxquels on peut s'attendre pour des technologies à 0,18µm ne sont plus que de quelques dixièmes de ns. Enfin, le temps de traversée des switches peut être réduit en réalisant les switches non pas avec des multiplexeurs, mais avec des circuits optimisés, « full-custom ». Il est clair que de tels switches donneraient des résultats nettement meilleurs que de simples multiplexeurs.

5.2.4 Interconnexions

Les interconnexions représentent un surplus matériel difficilement chiffrable, puisqu'on ne peut pas le mesurer en nombre de portes équivalentes, et que seule la comparaison entre deux circuits identiques fonctionnellement, l'un possédant les moyens de reconfiguration et l'autre pas est une solution satisfaisante de détermination de ce surplus. Ce travail est difficilement réalisable puisqu'il demande de pousser la réalisation des structures jusqu'à leur terme.

Qualitativement, la complexité du réseau d'interconnexion influe sur la complexité du circuit, et notamment sur le nombre de niveaux de métaux. Comme il a été spécifié dans le choix du réseau de reconfiguration locale, nous avons choisi de rendre le réseau aussi régulier que possible, sans chevauchement des interconnexions, de façon à limiter ce coût. Ainsi, celui-ci est plus faible que ceux des réseaux proposés dans la littérature. En terme de bande passante supplémentaire, le tableau 4 résume le surplus d'interconnexions des différentes structures et schémas proposés.

Structure de réseau	surplus d'interconnexions
Maillage normal , réseau local	200 %
Maillage avec interconnexions mono-directionnelles, réseau local	150 %
Maillage réduit, réseau local	50 %
Réseau global, colonnes supplémentaires uniquement	2*(nombre de lignes de blocs élémentaires)
Réseau global, lignes + colonnes supplémentaires	2*(nombre de lignes + nombre de colonnes de blocs élémentaires)

Tableau 4 : surplus d'interconnexions des différents schémas

5.3 Résultats de reconfiguration

5.3.1 Reconfiguration locale

5.3.1.1 Introduction

L'indépendance entre l'architecture et la méthode de reconfiguration a pour conséquence que la simulation a une importance primordiale pour déterminer les taux de reconfiguration.

Nous avons choisi de les mesurer par rapport à une ligne de processeurs supplémentaires : c'est la limite supérieure que nous nous sommes fixée en raison de la taille de nos blocs élémentaires. Un grand nombre de processeurs supplémentaires permet alors de mieux juger de l'efficacité de la méthode.

Par ailleurs, l'algorithme de placement n'est pas défini de façon unique par la méthode. Par conséquent, la reconfiguration peut s'avérer impossible avec un algorithme et possible avec un autre, les deux possédant les mêmes caractéristiques de qualité. Nous avons donc choisi de ne pas utiliser un seul algorithme de reconfiguration, mais une combinaison de plusieurs placements-routages.

Afin de pouvoir simuler les architectures sur des réseaux de taille assez conséquente, les placements que nous utilisons sont de deux types principaux : par blocs de 2 ou 3 lignes (voir chapitre 5) . Ils possèdent l'avantage de limiter la durée de routage tout en conservant une bonne qualité de reconfiguration, ceci pour un nombre de fautes important comme nous allons le montrer. Deux masques de placement différents ont été utilisés pour les blocs de 2 lignes, et un seul pour les blocs de 3 lignes. Nous avons ainsi obtenu 3 algorithmes de placement des processeurs que nous avons ensuite inversés horizontalement et/ou verticalement, afin d'obtenir 12 placements possibles. Enfin, afin d'assurer que le placement sera rapidement effectué, le nombre de boucles de chacun de ces algorithmes a été limité à 500, chiffre au-dessus duquel l'algorithme est dit en échec. Cette dernière précaution est prise afin de limiter fortement la durée de l'algorithme, car un grand nombre de cas doit être simulé.

Nous utilisons l'algorithme de routage présenté dans le chapitre 5. De la même façon que pour le placement, l'ordre de routage peut influer sur son efficacité. 4 algorithmes de routages sont donc utilisés, déduits du premier par inversion horizontale et/ou verticale de l'ordre de routage. Enfin, de même que pour le placement, le nombre de boucles de chaque algorithme est limité à 500.

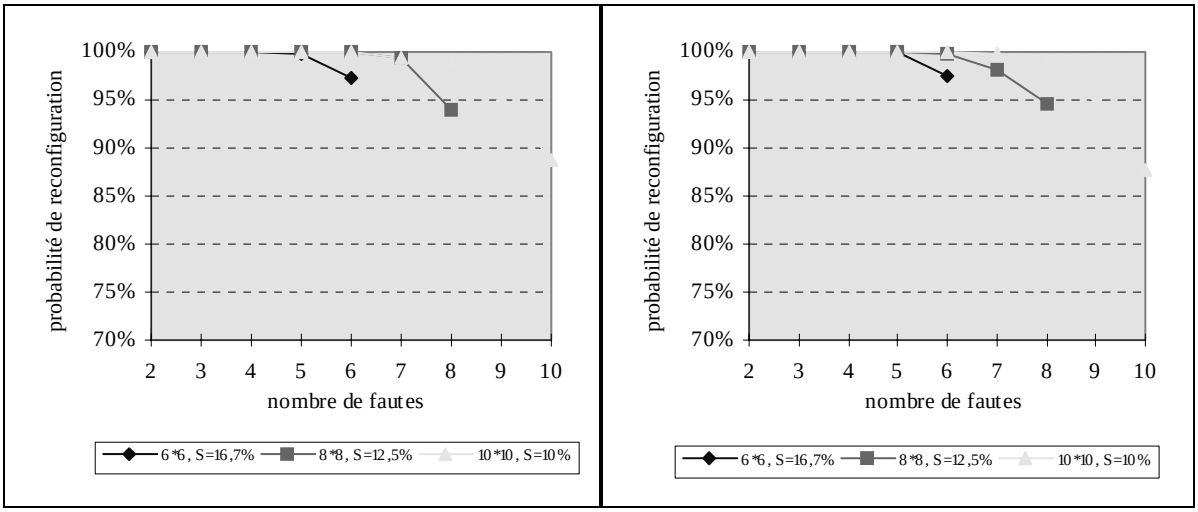
L'algorithme de reconfiguration est défini à partir de toutes les combinaisons de ces 12 placements et 4 routages. Il est alors utilisé pour le calcul des taux de reconfiguration locale.

5.3.1.2 Résultats de reconfiguration d'un réseau de grille 2-D

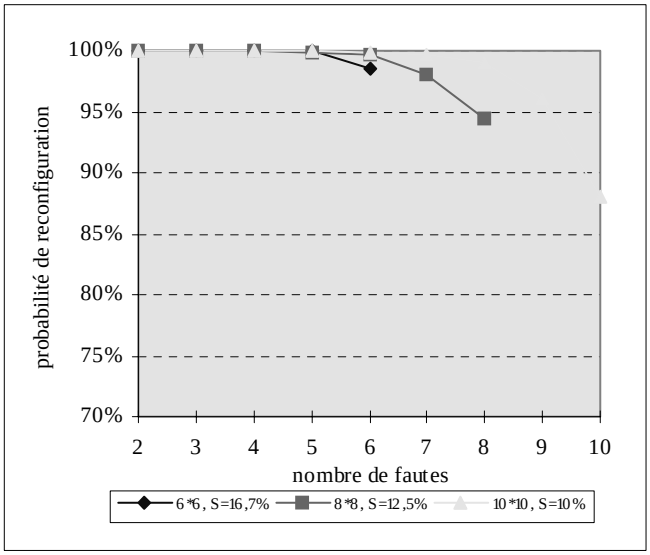
Les résultats de reconfiguration des réseaux en grille à liaisons mono-directionnelles (figures 13 et 14) comportant 150 % d'interconnexions supplémentaires sont présentés figure 18. Chaque architecture est évaluée pour 6*6, 8*8 et 10*10 processeurs. On a autorisé 4 switches dans les chemins de données entre deux processeurs voisins.

Ces résultats ont été obtenus ainsi :

- calcul de toutes les configurations possibles de processeurs fautifs pour déterminer le nombre de processeurs garantis reconfigurables ;
- au-delà : calcul de 10000 configurations de fautes et vérification de la stabilité des résultats obtenus (convergence).



(a) **(b)**



(c)

Figure 18 : taux de reconfiguration du réseau grille avec des communications mono-directionnelles. Les résultats (a), (b) et (c) correspondent respectivement aux structures des figures 13, 14a et 14b

Le nombre de processeurs garantis reconfigurables est de 4 pour chacune des structures 1,2 et 3. Ces structures diffèrent très peu en taux de reconfiguration. Elles diffèrent par contre par les configurations de fautes qu'elles sont capables de reconfigurer. Ainsi, une structure de type 1 peut être bloquée par une configuration de fautes qu'une structure de type 2 traitera facilement.

La figure 19 présente un exemple de reconfiguration obtenue par le simulateur. On peut remarquer que les interconnexions supplémentaires sont efficacement et largement utilisées.

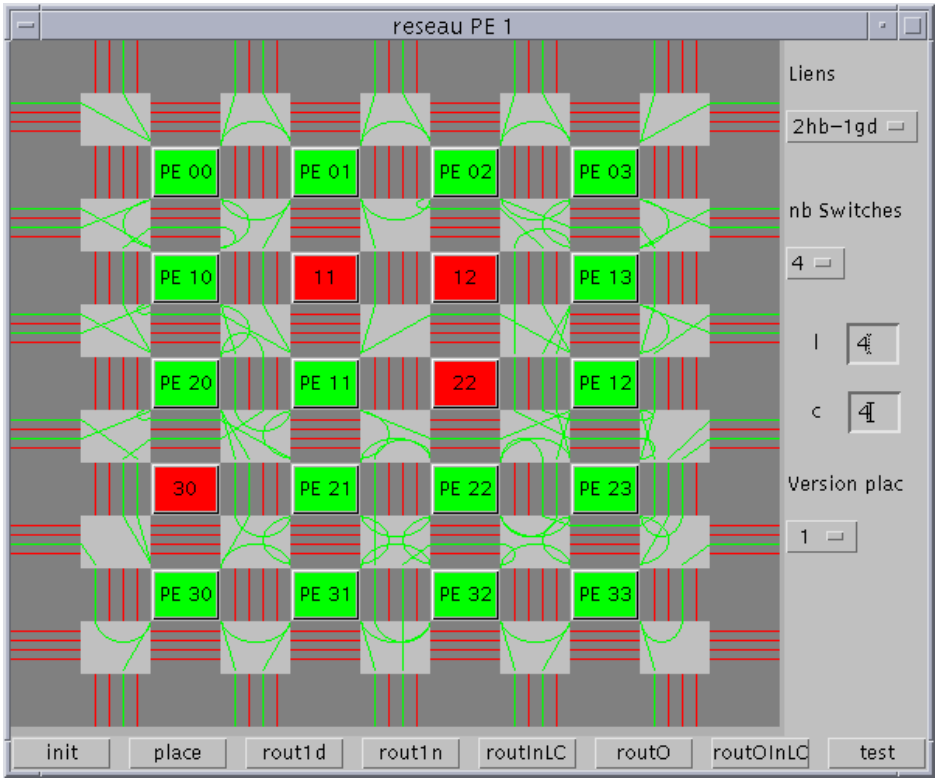


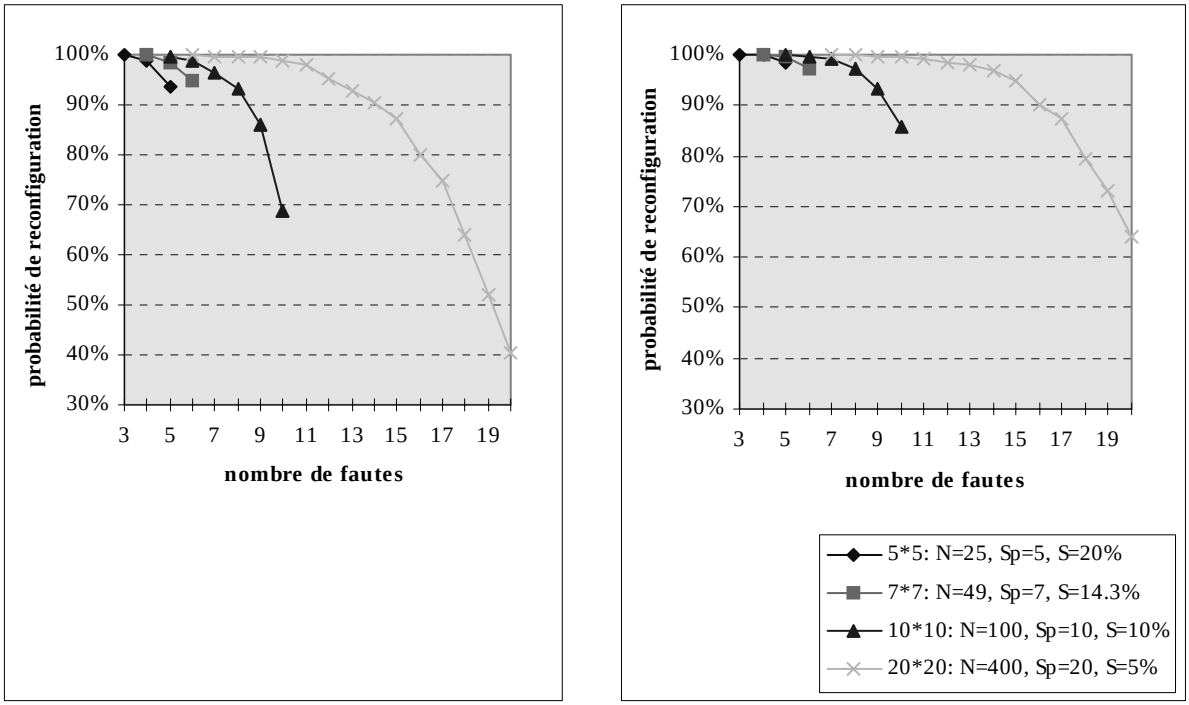
Figure 19 : prototype du réseau à liens mono-directionnels réduits correspondant à la figure 13. En vert , les processeurs fonctionnels. En rouge, les processeurs fautifs. Les traits verts représentent les interconnexions entre processeurs voisins

5.3.1.3 Résultats de reconfiguration d'un réseau réduit

La figure 20 présente les résultats de reconfiguration pour le réseau réduit du paragraphe 3.2, qui possède toutes les possibilités de communications décrites figure 16. Pour le réseau réduit, il est possible d'autoriser 4 ou seulement 3 switches dans les chemins de donnée entre deux processeurs voisins, ceci donnant des résultats différents. Ces résultats sont obtenus de la même façon que ceux du paragraphe précédent. Ils nous permettent d'assurer que 3 processeurs sont garantis reconfigurables pour un schéma à 3 switches, alors que 4 processeurs sont garantis reconfigurables pour un schéma à 4 switches.

Les résultats de la figure 20 montrent que la reconfiguration avec seulement 3 switches autorisés dans les chemins de communications est très correcte. Les capacités de nos switches sont donc très bonnes et servent effectivement à limiter les interconnexions supplémentaires.

La figure 21 présente un exemple de reconfiguration obtenue par le simulateur, pour la configuration de la figure 16 b.



(a) **(b)**

Figure 20 : taux de reconfiguration du réseau réduit comportant six configurations de communication différentes. (a) avec 3 switches autorisés dans le chemin de données. (b) avec 4 switches autorisés dans le chemin de données. N = nombre de processeurs du bloc élémentaire, Sp = nombre de processeurs supplémentaires, S = surplus matériel dû aux processeurs supplémentaires

5.3.1.4 Conclusion

Les résultats de reconfiguration, aussi bien pour le réseau maillage que pour le réseau réduit, sont de bonne qualité. Ils sont comparables aux autres schémas de la littérature qui comportent, pourtant, pour les plus efficaces, au moins une ligne et une colonne de processeurs supplémentaires. Ainsi, la méthode de reconfiguration proposée est la seule de la littérature à mettre en avant une très grande modularité et des taux de reconfiguration très bons pour des structures comportant un nombre faible de processeurs supplémentaires. D'autre part, la structure très régulière, sans aucun croisement de connexions, est un atout indéniable que les autres structures ne possèdent pas. La reconfiguration proposée est donc tout à fait adaptée au cas des calculateurs fortement intégrés et fortement couplés.

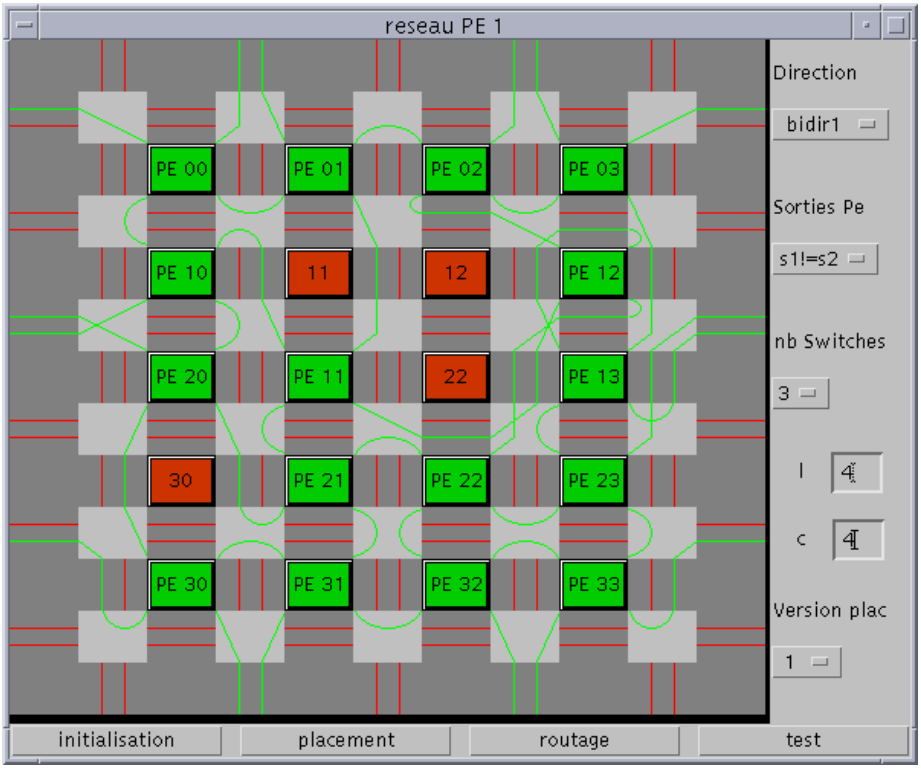


Figure 21 : prototype du réseau réduit. En vert, les processeurs fonctionnels. En rouge, les processeurs fautifs. Les traits verts représentent les interconnexions entre processeurs voisins

5.3.2 Reconfiguration globale

La reconfiguration globale s'effectue sur des lignes ou des colonnes de blocs élémentaires. Son efficacité est donc forcément réduite, et elle ne pourra pas être utilisée avec un bon rendement de reconfiguration (nombre de fautes reconfigurables par rapport au surplus matériel engendré). Toutefois, elle permet de résoudre les problèmes que sont les fautes dans les interconnexions entre blocs élémentaires et les groupements de fautes importants dans un même bloc élémentaire.

Pour ces raisons, les schémas utilisant une reconfiguration globale n'utilisent pas plus d'une ou deux lignes ou colonnes supplémentaires. Nous nous contentons donc d'indiquer les taux obtenus pour une ou deux colonnes supplémentaires (figure 22). Il est à noter que ces taux ne dépendent absolument pas de la taille des blocs élémentaires, mais uniquement du nombre de blocs élémentaires dans le réseau complet. La comparaison entre le schéma avec une ou deux colonnes supplémentaires donne un net avantage au deuxième cas en terme d'amélioration de la fiabilité. Toutefois, le surplus matériel engendré ne peut être toléré que pour les très fortes demandes en fiabilité.

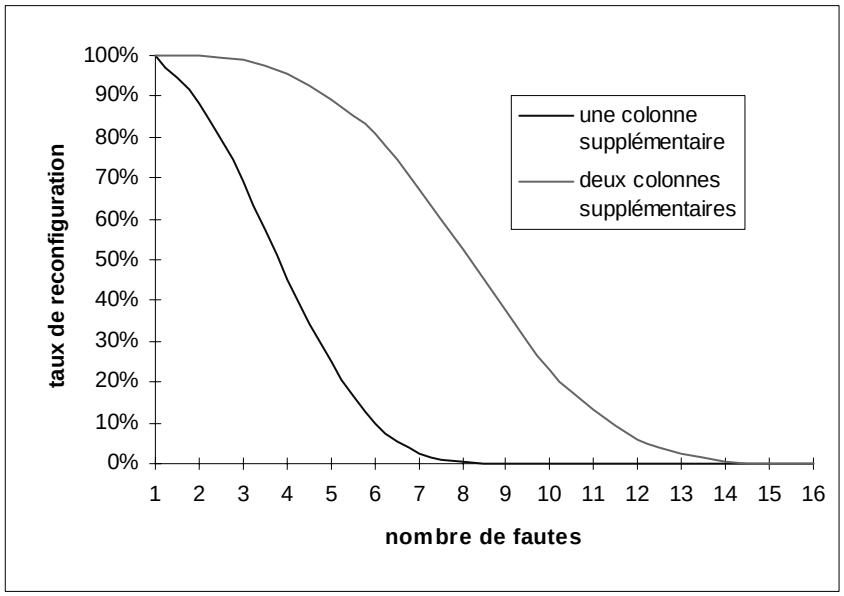


Figure 22 : taux de reconfiguration pour un réseau de 64 ASIC de 16 processeurs chacun utilisant une reconfiguration globale avec une ou deux colonnes de processeurs supplémentaires

5.4 Conclusion

Les résultats concernant le surplus matériel, les pertes en performance, le surplus en terme de bande passante et les taux de reconfiguration ont été présentés. Nous disposons donc maintenant de la plupart des éléments nécessaires au choix de la structure tolérante aux fautes. Il nous manque cependant le lien entre les taux de reconfiguration et la fiabilité. Ce lien, qui nous donne la clé du choix du nombre de processeurs supplémentaires, est maintenant présenté.

6 Amélioration de la fiabilité

6.1 Introduction

Le taux de reconfiguration d'un système ne représente pas de façon explicite l'augmentation de fiabilité. Ainsi, dans cette section, nous étudions la fiabilité par rapport au schéma de reconfiguration locale uniquement. Ce choix est justifié par le fait que la reconfiguration globale sert uniquement, dans notre cas, à traiter des fautes d'interconnexions et des amas de fautes. Nous supposons également que les fautes surviennent dans les processeurs. Enfin, nous supposons que la répartition des fautes dans les circuits de la structure complète est uniforme.

Avant d'introduire le calcul de la fiabilité, il nous paraît nécessaire de replacer les fautes garanties reconfigurables dans le contexte de la fiabilité. La fiabilité R(t) d'un composant ou d'une structure est la *probabilité* d'atteindre un temps t de fonctionnement. Il est donc possible que le temps véritable de fonctionnement soit bien en dessous (ou au-dessus) de cette valeur. Le fait de garantir la reconfiguration ne garantit pas une durée de fonctionnement minimale. Il n'est donc pas utile de *garantir* la reconfiguration à condition que la probabilité d'atteindre un temps de fonctionnement reste importante.

Cela nous permet d'introduire une discussion sur le bien-fondé du MTTF. Ce « temps moyen » jusqu'à une panne représente l'*espérance* mathématique de la densité de défaillance. Or, en terme de probabilité, cette valeur n'a un sens que si elle est associée à la variance, qui représente l'écart moyen autour de l'espérance. Cette valeur n'est généralement pas fournie dans la littérature, ce qui pose des problèmes de clarté des résultats. Dans la suite, après avoir éclairci la notion de MTTF, nous évaluons théoriquement l'amélioration en fiabilité R(t) apportée par le schéma de reconfiguration locale. Cette étape nous permet de valider un simulateur de type Monte Carlo qui nous permet d'obtenir non seulement le temps moyen de fonctionnement, mais également la répartition de ce temps, et donc les probabilités précises d'atteindre une durée de vie fixée. Nous verrons également que ce simulateur introduit des possibilités intéressantes, comme le calcul de la fiabilité sous contrainte thermique ou sous l'effet de l'insertion d'un « point froid ».

6.2 Signification du MTTF

La notion de MTTF est généralement utilisée pour la mesure de la fiabilité, mais sa signification physique est rarement présentée. Nous allons donc éclaircir ici cette notion importante.

Afin de se donner tous les éléments de compréhension, nous allons repartir de données qui ont été présentées dans le chapitre 1. Nous supposons donc tout d'abord que le taux de répartition des fautes Z(t) de chaque processeur suit une courbe en baignoire, et nous nous plaçons durant la période dite de vie utile pendant laquelle Z(t) est une constante notée λ et appelée taux de défaillance. Dans ces conditions, $R(t) = e^{-\lambda t}$.

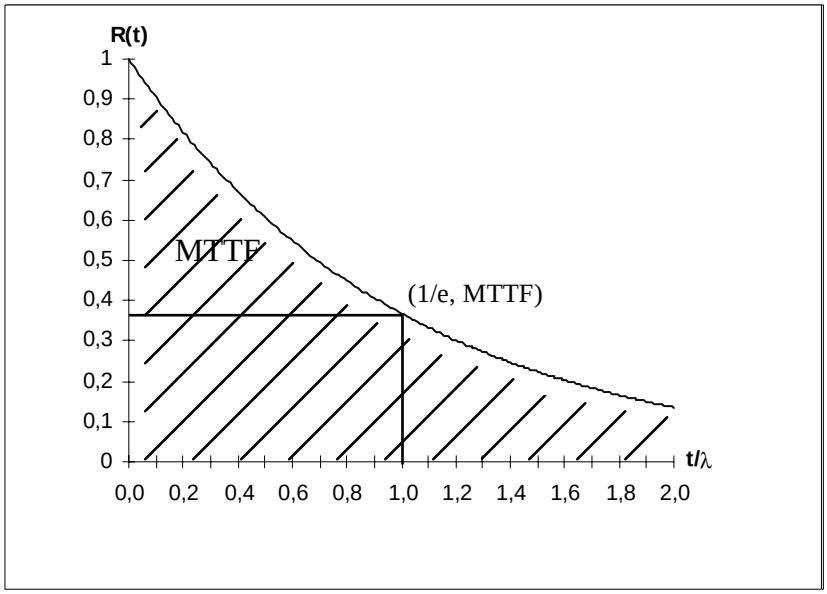


Figure 23 : courbe $R(t) = e^{-\lambda t}$ en fonction de t/λ. Le MTTF est calculé par la surface délimitée par R(t) et correspond à une probabilité de fonctionnement de 1/e

Le MTTF du système est l'espérance mathématique de la densité de défaillance, soit $MTTF = \int_0^\infty R(t)dt$, c'est à dire que le MTTF représente la surface délimitée par la courbe R(t) (figure 23). La résolution de l'équation précédente donne MTTF = 1/λ, ce qui correspond à $R(t) = 1/e \cong 0.37$.

Cela signifie que le système a environ 37 % de chances de fonctionner encore correctement au bout du temps MTTF.

Le MTTF ne représente donc pas un temps « garanti » de bon fonctionnement, mais une moyenne de tous les temps de fonctionnement possibles. Si on veut avoir une certaine garantie de bon fonctionnement, il est alors plus judicieux de prendre le temps qui correspond, par exemple, à 99 % de chances de bon fonctionnement. Par conséquent, donner le MTTF seul n'a pas un sens très précis en terme de fiabilité et on préférera la courbe R(t) ou tout au moins une approximation de cette courbe. Nous conserverons néanmoins cette notion de MTTF pour la comparaison des différentes structures de reconfiguration.

6.3 Calcul de la fiabilité d'un composant ou d'un réseau à reconfiguration globale idéale

Le calcul de la fiabilité d'un composant à reconfiguration locale revient à celui d'un réseau à reconfiguration globale à grain fin. Dans toute la suite de cette section, la reconfiguration est supposée comme étant idéale, c'est à dire que si le système comprend S éléments supplémentaires, on suppose que les S éléments peuvent remplacer n'importe quelle configuration de S fautes.

Etant donné que le taux de défaillance d'un processeur est constant pendant la durée de vie utile, la loi de distribution des fautes est une loi uniforme. Ainsi, pour le calcul du taux de répartition des défaillances, nous définissons Z zones ou intervalles de fiabilité de taille égale, leur réunion formant l'espace des probabilités, c'est à dire l'intervalle [0,1].

Si le système comporte N éléments, le calcul de la répartition de la fiabilité revient au calcul de la probabilité que la S+1^{ième} faute, qui entraîne une défaillance du système, se trouve dans la i^{ième} zone. Ce calcul est donc un problème de dénombrement de tous les cas, le nombre de zones Z donnant la précision du calcul. Le problème de dénombrement est soumis à une explosion combinatoire de type factoriel, et ce calcul de fiabilité est donc limité à des réseaux de taille très limitée. Il nous sert néanmoins à valider le simulateur qui est présenté par la suite (paragraphe 6.6).

6.4 Calcul de la fiabilité d'un réseau à reconfiguration locale idéale

Dans cette section, nous supposons que le système est composé de N blocs, chacun de ces blocs pouvant tolérer à coup sûr S fautes (et comportant donc S processeurs supplémentaires). Le calcul de la fiabilité revient alors à celui de la probabilité d'apparition de chaque événement entraînant une panne du système.

Afin d'illustrer notre propos, prenons le cas de 3 ASIC de 8 processeurs pouvant tolérer chacun 1 faute. La panne du système survient lorsque deux processeurs sont fautifs dans un même ASIC. Nous pouvons alors construire l'arbre d'apparition des événements en leur adjoignant une probabilité. Dans une première approximation, nous supposons que les probabilités d'apparition d'une faute dans les blocs sont identiques quel que soit l'état précédent de la structure.

Par exemple, à partir de la configuration 1.0.0 (une faute dans le premier ASIC et 0 faute dans les 2 suivants), on peut arriver à la configuration 1.1.0 avec une probabilité de 2/3 et à la configuration 200 avec une probabilité d'1/3. La figure 24 nous donne alors l'arbre complet pour ce cas.

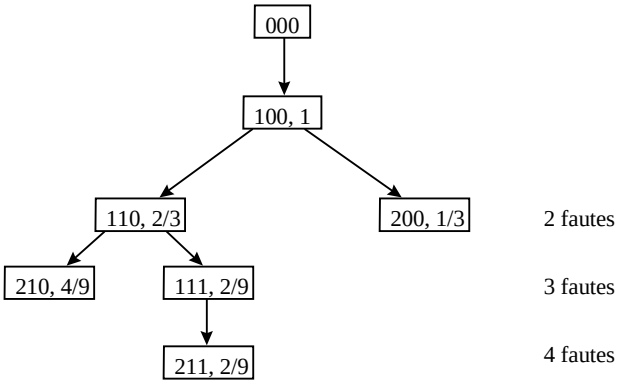


Figure 24 : arbre de fautes pour 3 blocs de 8 processeurs comportant chacun un processeur supplémentaire

En traduisant les probabilités en pourcentages, on obtient les résultats suivants :

- 33,33 % des structures sont en panne lorsque 2 processeurs sont fautifs ;
- 44,44 % des structures sont en panne lorsque 3 processeurs sont fautifs ;
- 22,22 % des structures sont en panne lorsque 4 processeurs sont fautifs.

Cette première approximation permet d'obtenir un histogramme simplifié de la fiabilité du système. Ce résultat est toutefois imprécis. En effet, nous avons supposé que chaque bloc présente, à chaque événement, la même probabilité qu'une nouvelle faute se produise. Or ceci est faux, puisqu'un bloc dont un des processeurs est fautif a une probabilité plus faible d'apparition de nouvelle faute, puisqu'il comporte moins de processeurs. Un coefficient correcteur est alors nécessaire pour prendre en compte les différences de probabilité d'apparition de faute. Il est d'autant plus faible que le nombre de processeurs par bloc est grand par rapport au nombre de processeurs supplémentaires.

Si nous reprenons l'exemple précédent, nous avons N=27 processeurs dans le système, et N_b=9 processeurs par bloc au départ. Pour une configuration de fautes, les probabilités d'apparition de fautes p(i) pour chaque bloc i est :

$$p(i) = \frac{N_b - nf(i)}{N - nf} \qquad (1)$$

où nf et nf(i) sont, respectivement, le nombre de fautes du système et du bloc i ($nf = \sum_i nf(i)$).

Par exemple, pour la configuration 1.0.0, les probabilités d'apparition de faute pour chaque bloc sont p(1)=0,31, p(2)=p(3)=0,345.

Il est donc nécessaire de refaire l'arbre de la figure 24 en tenant compte de cette correction, ce qui revient à considérer l'ordre d'apparition des événements. Les nouvelles répartitions sont données figure 25.

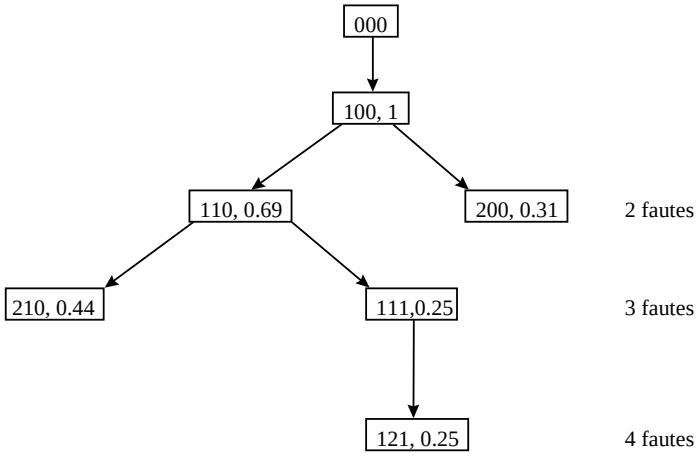


Figure 25 : probabilités de répartition des fautes rectifiées, pour 3 blocs de 9 processeurs dont un supplémentaire

Cet exemple nous montre que l'effet peut être assez important, l'état de panne étant atteint pour 4 fautes dans 25 % des cas au lieu de 22,2 %, soit une erreur de plus de 10 %.

Une remarque importante concerne la réalisation de l'arbre et sa simplification. En effet, le calcul de chaque branche de l'arbre est inutile. Ainsi, deux remarques peuvent nous aider à simplifier sa réalisation.

Premièrement, la place des fautes dans les blocs n'a pas d'importance, chaque bloc ayant la même probabilité de se reconfigurer : par exemple, les configurations 110 et 101 ont les mêmes probabilités d'apparition.

La seconde remarque vient du calcul des probabilités. En effet, la probabilité d'atteindre un état est indépendante du chemin parcouru (c'est à dire de l'ordre d'apparition des événements). Prenons l'exemple de 3 blocs comportant chacun 8 processeurs utiles et 2 processeurs supplémentaires. Les chemins (1.0.0 → 2.0.0 → 2.1.0) ou (1.0.0 → 1.1.0 → 2.1.0) sont tous les deux possibles pour arriver à l'état de fautes 2.1.0. Pour le premier de ces chemins, la probabilité d'arriver à l'état 2.1.0 est : $\frac{9}{29} * \frac{10}{28}$, alors que dans le second cas, cette probabilité est : $\frac{10}{29} * \frac{9}{28}$. Le facteur correctif est donc le même pour tous les chemins menant aux mêmes configurations de fautes. Le calcul théorique revient donc à dénombrer tous les chemins menant à une configuration de fautes, puis à appliquer le facteur correctif.

Ce calcul nous permet alors d'obtenir l'histogramme de répartition des fautes, c'est à dire la probabilité que le système tolère de une à N fautes, N correspondant à l'ensemble de tous les processeurs supplémentaires de la structure.

A partir de cet histogramme, il est possible de se rapporter au calcul de la valeur moyenne d'occurrence d'une faute, c'est à dire au calcul du temps moyen de bon fonctionnement ou MTTF. Pour cela nous utilisons la courbe de fiabilité $R(t) = e^{-\lambda t} = e^{-T}, T = \frac{t}{\lambda}$ (nous nous plaçons par rapport au temps de fonctionnement moyen d'un processeur). La loi de répartition des fautes étant uniforme, lorsqu'une structure comportant N processeurs en tout est en panne après l'apparition de f fautes, la durée de vie moyenne est : $T_f = -\ln(\frac{f}{N} + \frac{1}{2*N})$. Le calcul de la durée de vie moyenne du système revient alors à associer la durée de vie moyenne pour chaque faute f à la probabilité de tolérer f fautes, donnée par l'histogramme précédent, c'est à dire :

$$MTTF_{Structure} = \sum_{f=0}^{f=S} p(f) * T_f = -\sum_{f=0}^{f=S} p(f) * \ln(\frac{f}{N} + \frac{1}{2*N}) \qquad (2)$$

S étant le nombre total de processeurs supplémentaires de la structure et N le nombre total de processeurs de la structure.

Nous en déduisons ainsi le temps moyen de bon fonctionnement d'une structure tolérante aux fautes par rapport au temps moyen de bon fonctionnement d'un processeur de cette structure.

En procédant au regroupement des branches dans le modèle d'arbre retenu, il est possible de traiter des structures de faible taille ou comportant 1 ou 2 processeurs supplémentaires par bloc. Cette modélisation est donc limitée et ne permet pas une évolution vers d'autres calculs de fiabilité. Pour cette raison, un simulateur de Monte Carlo a été conçu.

6.5 Présentation du simulateur

Les calculs théoriques limitent les possibilités d'extension du modèle de fiabilité. Afin de pouvoir intégrer d'autres paramètres influant sur la fiabilité, comme les phénomènes de température, un simulateur par Monte Carlo a été conçu. La simulation est basée sur plusieurs étapes :

1. Tirage des fiabilités R(t) des N processeurs de la structure en fonction des contraintes extérieures ;
2. Classement de ces fiabilités en ordre décroissant et transformation en temps d'apparition de fautes (normalisé par rapport au processeur) ;
3. Pour chaque pas de temps, tirage du bloc dans lequel apparaît la faute en prenant en compte les contraintes sur les blocs ;
4. Si la faute est reconfigurable, on passe à la faute suivante. Sinon, le temps obtenu est le temps de bon fonctionnement de l'échantillon.

Le critère de convergence est défini comme étant l'écart entre un nombre déterminé E d'échantillons (c'est à dire de scénarii d'apparition de fautes) et la moyenne de tous les échantillons précédents. Des séries de E scénarii sont alors tirés. Pour chacun, on calcule les répartitions de durée de vie. L'algorithme s'arrête alors dès que le critère de convergence passe sous une limite fixée (par exemple, inférieur à 0.001).

Ce simulateur permet le calcul de la fiabilité (histogramme de fiabilité et MTTF) du système par rapport à celle d'un processeur pour une structure à reconfiguration locale. La reconfiguration globale à grain fin est alors vue comme un cas particulier comportant un seul bloc élémentaire.

Les avantages de ce simulateur sont, outre une durée de calcul limitée, la possibilité de prendre en compte des facteurs physiques, comme la température de chaque partie d'un ASIC en fonction du positionnement des processeurs dans cet ASIC. Il est ainsi possible de calculer l'effet d'un point chaud au niveau de la fiabilité, mais également celui de l'insertion d'un « point froid », c'est à dire un processeur supplémentaire non cadencé (arbre d'horloge coupé pour ce processeur). Ce dernier point, permettant une augmentation de la fiabilité du bloc, fait partie des perspectives envisagées pour l'extension de ce travail.

La section suivante traite des résultats obtenus par le simulateur présenté ci-dessus ainsi que du simulateur de calcul théorique. A l'issue de la présentation des résultats, nous nous poserons la question fondamentale du choix entre la reconfiguration à grain fin locale et à grain fin globale.

6.6 Résultats et discussion

6.6.1 Introduction

Afin de présenter les résultats de manière cohérente et claire, nous supposerons que toutes les configurations de fautes sont reconfigurables, quelle que soit la taille du réseau. Il est clair que cette hypothèse avantage les structures où un nombre important de processeurs peut être remplacé. Ainsi, la reconfiguration globale à grain fin est avantagée par rapport à la reconfiguration locale.

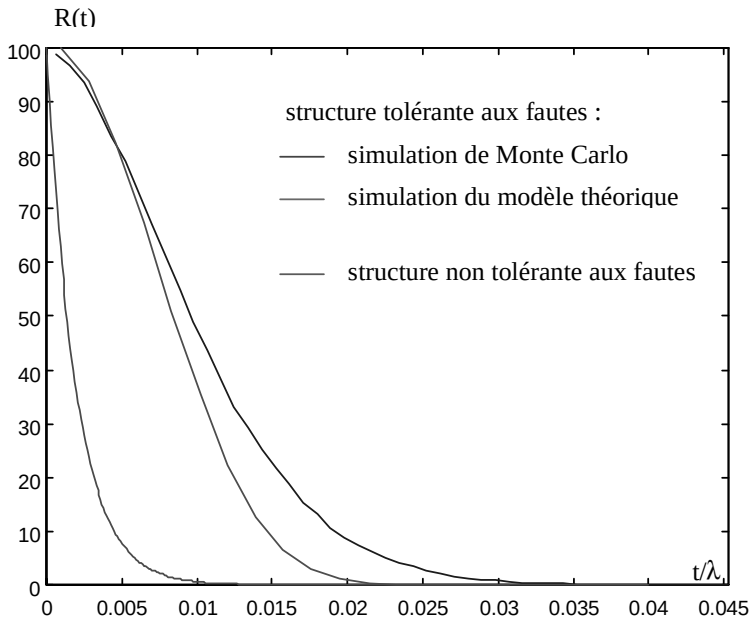
Dans cette section, les résultats présentant les courbes de fiabilité R(t) sont tout d'abord donnés. Nous comparons alors les évolutions des courbes de fiabilité en fonction des processeurs supplémentaires pour plusieurs structures physiques. Ces résultats sont présentés pour les simulateurs théoriques et de Monte Carlo, ce qui nous permet de les comparer.

Nous nous basons ensuite sur le MTTF pour donner plus précisément et de façon plus lisible l'augmentation de fiabilité apportée par chacun des schémas, et une idée du niveau de fiabilité qu'il est possible d'atteindre.

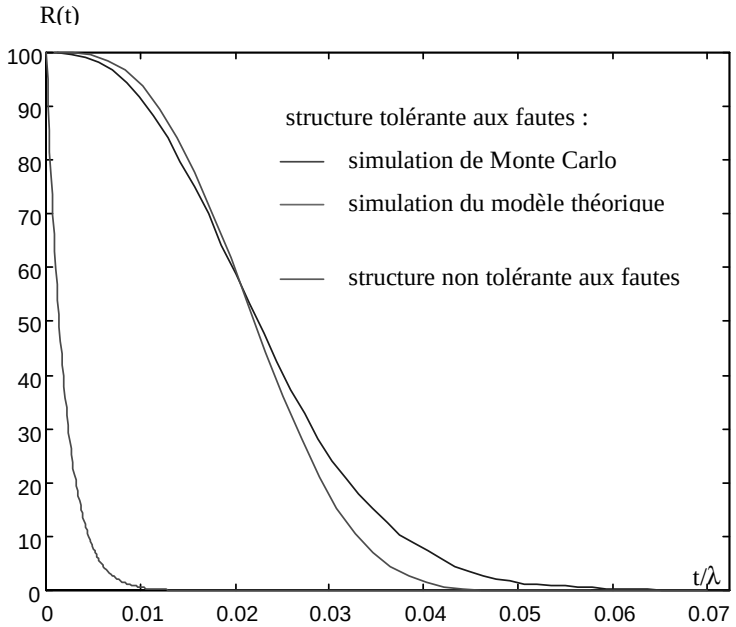
Enfin, dans une troisième partie, les schémas de reconfiguration à grain fin globale et locale sont comparés.

6.6.2 Calcul de R(t)

Dans cette section, nous allons nous intéresser aux courbes de fiabilité R(t) obtenues par le simulateur théorique et le simulateur de Monte Carlo. Pour cela nous allons prendre un exemple de réseau composé de 16 ASIC de 32 processeurs fonctionnels chacun, soit 512 processeurs.



(a)



(b)

Figure 26 : courbes de fiabilité R(t) pour un réseau de 16 ASIC de 32 processeurs, la reconfiguration étant locale au niveau de l'ASIC, avec (a) 1 processeur supplémentaire par ASIC et (b) 2 processeurs supplémentaires par ASIC

La figure 26 donne les résultats obtenus avec les deux simulateurs. Ils sont comparés avec la courbe théorique d'un système non tolérant aux fautes. Le modèle théorique est moins optimiste que la simulation de Monte Carlo. Cette différence peut être expliquée par le calcul des temps de

fonctionnement : pour le calcul théorique, nous calculons une moyenne en linéarisant la courbe R(t) d'un processeur, ce qui entraîne des erreurs. Ainsi, avec le modèle théorique, il est impossible qu'un circuit dépasse une durée de vie fixée par le nombre de processeurs supplémentaires disponibles, alors que dans la réalité, ce cas, bien que très rare, puisse parfaitement se produire. L'erreur produite dépend alors du nombre de processeurs supplémentaires du système, ce qui explique qu'elle est moins importante pour le cas avec 2 processeurs supplémentaires par ASIC que pour le cas avec un seul processeur supplémentaire par ASIC.

La figure 27 donne les résultats de fiabilité R(t) obtenus par Monte Carlo pour une structure de 16 ASIC comprenant de 0 à 4 processeurs supplémentaires par ASIC. On voit nettement l'augmentation de fiabilité obtenue par le schéma avec seulement un processeur supplémentaire par ASIC par rapport au schéma sans tolérance aux fautes.

L'allure de la courbe de fiabilité change également, et la chute de fiabilité observée dans le système non tolérant aux fautes n'est plus aussi importante dans les structures avec reconfiguration locale. Par comparaison, la courbe de fiabilité obtenue avec une reconfiguration globale à grain fin qui permettrait de reconfigurer n'importe quelle configuration de fautes jusqu'à 32 fautes est également montrée sur ce graphique. Cette courbe montre une meilleure amélioration de la fiabilité avec une courbe de descente plus raide, et donc de meilleurs résultats pour les hautes probabilités de bon fonctionnement. En terme de fiabilité seule, le schéma à reconfiguration globale à grain fin est donc meilleur que celui à reconfiguration locale.

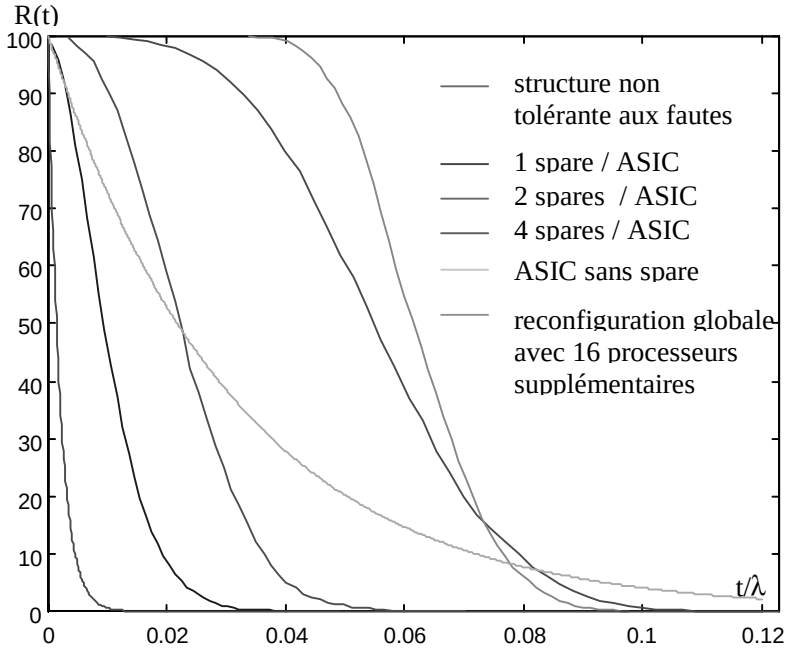


Figure 27 : courbes de fiabilité R(t) obtenues par simulation de Monte Carlo pour une structure composée de 512 processeurs répartis en 16 ASIC de 32 processeurs chacun

Une autre comparaison intéressante peut être faite avec la courbe de fiabilité d'un ASIC (figure 27). Un objectif d'amélioration de la fiabilité cohérent peut être de vouloir atteindre, pour la structure complète, une fiabilité équivalente à celle d'un ASIC. Dans le cas de la structure en exemple, un seul processeur supplémentaire par ASIC est alors suffisant pour assurer plus de 90 % de probabilité de bon fonctionnement. De même, au-delà de 50 % de probabilité de bon fonctionnement, la structure avec 2 processeurs supplémentaires par ASIC est nettement meilleure qu'un ASIC seul. Par conséquent, la seule comparaison des MTTF de ces structures biaise l'amélioration réelle de fiabilité.

Dans le paragraphe suivant, nous nous ramenons toutefois à la notion de MTTF, que nous utilisons afin de pouvoir comparer facilement les différents schémas de reconfigurations locale et globale à grain fin. Nous nous plaçons donc dans un cas défavorable, l'amélioration de fiabilité réelle étant meilleure que celle indiquée.

6.6.3 Amélioration du temps moyen de bon fonctionnement

Le tableau 5 présente des comparaisons de réseaux logiques identiques en terme de nombre de processeurs (1024) et de processeurs supplémentaires (128), mais de structures physiques différentes. Il nous montre clairement que plus la reconfiguration est globale (dans le cas présenté ici, moins il y a d'ASIC), plus les résultats sont bons. Ainsi, l'incidence du nombre de processeurs par ASIC est très importante en terme d'augmentation de fiabilité. Les chiffres de résultats de durée de vie moyenne plaident toutefois en faveur de la modularité de la reconfiguration. En effet, dans beaucoup d'applications non critiques, il est inutile que la durée de vie moyenne d'une structure dépasse celle d'un des composants. L'amélioration de fiabilité, même dans la première structure, est donc bonne.

N_{ASIC}	$N_{PE/ASIC}$	$N_{Sp/ASIC}$	$\frac{MTTF_{ASIC}}{MTTF_{PE}}$	$\frac{MTTF_{ST}}{MTTF_{PE}}$	$\frac{MTTF_{ST}}{MTTF_{SNT}}$	durée de vie moyenne
64	16	2	0,0625	0,027	27,7	6,5 ans
32	32	4	0,031	0,046	47,2	22,3 ans
16	64	8	0,015	0,067	68,4	67 ans

Tableau 5 : résultats pour une structure de 1024 processeurs plus 128 processeurs supplémentaires. ST = Structure Tolérante aux fautes, SNT = Structure Non Tolérante aux fautes. La durée de vie moyenne est calculée par rapport au MTTF d'un ASIC de 15 ans

Le tableau 6 se place dans une autre situation, c'est à dire une situation où le nombre de processeurs par ASIC est donné. C'est une situation réaliste, où la complexité des processeurs limite leur nombre dans un seul ASIC. Nous supposons ici que chaque ASIC comprend 32 processeurs fonctionnels. Notre objectif est alors de mesurer l'amélioration du MTTF par une reconfiguration locale avec 1,2 ou 4 processeurs supplémentaires par ASIC et par une reconfiguration globale à grain fin, ceci pour deux structures de 1024 et 4096 processeurs.

Le premier résultat est très intéressant, puisque le fait d'ajouter un seul processeur supplémentaire par ASIC permet de multiplier par plus de 7 la durée de fonctionnement de la structure à 1024

processeurs, et par plus de 14 la durée de fonctionnement pour la structure à 4096 processeurs pour seulement 3 % de processeurs supplémentaires.

La reconfiguration locale comportant un seul processeur supplémentaire par bloc montre également de bonnes qualités lorsqu'on augmente le nombre d'ASIC. En effet, entre 32 et 128 ASIC, la fiabilité est divisée par 4 pour une structure sans tolérance aux fautes alors que la durée de vie moyenne de la structure tolérante aux fautes n'est que divisée par 2.

Ce tableau nous permet également de montrer que de très bons résultats sont obtenus avec seulement 2 processeurs supplémentaires par ASIC, même pour des réseaux de taille importante. Enfin, la comparaison avec une reconfiguration globale montre que cette dernière reste meilleure au niveau du rapport entre le pourcentage de matériel ajouté et l'amélioration de la fiabilité (à condition, bien sûr, que toutes les configurations de fautes soient reconfigurables). La section suivante va au-delà de la seule amélioration en fiabilité et discute des avantages et inconvénients des méthodes de reconfiguration à grain fin locales et globales.

N_{PE}	N_{ASIC}	$N_{Sp/ASIC}$	% Sp	$\frac{MTTF_{ST}}{MTTF_{SNT}}$	durée de vie moyenne
1024	32	1	3 %	7,6	3,4 ans
1024	32	2	6 %	18,7	8,7 ans
1024	32	4	12,5 %	46,9	21,8 ans
1024	32	reconfiguration globale, 32 sp.		32,5	15,5 ans
4096	128	1	3 %	14,5	1,7 ans
4096	128	2	6 %	43,9	5,3 ans
4096	128	4	12,5 %	132,1	15,5 ans
4096	128	reconfiguration globale, 128 sp		126,6	14,8 ans

Tableau 6 : résultats pour des structures composées d'ASIC de 32 processeurs fonctionnels. ST = Structure Tolérante aux fautes, SNT = Structure Non Tolérante aux fautes. La durée de vie moyenne est calculée par rapport au MTTF d'un ASIC de 15 ans

6.6.4 Reconfiguration à grain fin : globale ou locale ?

Si le surplus matériel et l'augmentation de fiabilité sont les seules variables prises en compte, nous avons vu à plusieurs reprises que la reconfiguration globale était meilleure.

Néanmoins, d'autres facteurs peuvent intervenir et peuvent être prépondérants dans le choix. La limitation du nombre d'entrées/sorties d'un ASIC en est une, qui nous a fait préférer le choix de la reconfiguration locale. D'autres facteurs plaident en faveur de celle-ci.

En premier lieu, la reconfiguration que nous proposons est modulaire, contrairement aux schémas classiques de reconfiguration globale que nous avons vu au chapitre 4. Ainsi, limiter à 32 le nombre de processeurs supplémentaires pour un réseau de 1024 éléments n'est pas possible avec ces schémas : ils proposent au minimum 64, voire 128 processeurs supplémentaires. Ceux-ci amènent alors à une amélioration de fiabilité qui va bien au-delà des besoins de nos structures.

D'autre part, la reconfiguration locale n'oblige pas un arrêt total de la structure lors des phases de reconfiguration, ce qui peut s'avérer intéressant dans certains cas. De même, en cas d'échec de reconfiguration d'un ASIC, la structure peut continuer à fonctionner dans un mode dégradé. A ces avantages, s'ajoute également la rapidité de la reconfiguration locale.

Par ailleurs, les structures sur lesquelles s'appuie la reconfiguration locale sont plus simples que celles de la reconfiguration globale. Ceci est particulièrement vérifié dans le cas ou un seul processeur supplémentaire est ajouté par ASIC. Cette simplicité peut alors autoriser un fonctionnement du réseau pour d'autres communications « exotiques » tout en conservant la même fiabilité.

Enfin, et contrairement à la reconfiguration globale, notre reconfiguration locale est complètement transparente pour la conception de la structure à partir des ASIC, les entrées/sorties et fonctionnalisés de chaque ASIC étant inchangées.

Pour conclure, on peut remarquer que la reconfiguration locale devient obligatoire pour des structures de type Multi-SIMD formées de sous-réseaux SIMD. Ainsi, nous avons mis en exergue les avantages d'une reconfiguration locale à grain fin par rapport à son homologue globale, beaucoup moins souple.

6.6.5 Conclusion

Nous avons présenté ici les résultats d'augmentation de fiabilité en nous ramenant à un cas réaliste de structure fortement intégrée. Pour de telles structures, les besoins en fiabilité sont importants et nous avons vu que des schémas de reconfiguration locale pouvait les résoudre. Aussi, les résultats présentés ici donnent les moyens d'évaluer la fiabilité et de choisir le schéma de reconfiguration. Enfin, les avantages de la reconfiguration locale par rapport à la reconfiguration globale ont été mis en avant : celle-ci prend plus en compte les problèmes liés à la réalisation de la structure et aux limitations de la technologie, donnant un côté plus réaliste à la recherche d'augmentation de fiabilité.

6.7 Conclusion et perspectives

Dans cette section, ont été présentés deux simulateurs, l'un se basant sur un modèle théorique, et le second sur des tirages de type Monte Carlo. Les essais montrent une adéquation correcte de ces deux modèles, tout en mettant en exergue les limitations du modèle théorique. Le simulateur de Monte Carlo ayant ainsi été validé, il est maintenant possible de poursuivre le travail par une étude de l'effet de la température sur le système, et celui de l'adjonction de points froids. Pour cela, les caractéristiques physiques des ASIC doivent être rentrées, et cela constitue une de nos perspectives.

Dans la suite de la section, les résultats d'augmentation de fiabilité ont été donnés, permettant de répondre à la question de fiabilité posée dans l'introduction de ce mémoire. Il est ainsi montré que le

schéma de reconfiguration local permet de répondre aux besoins en fiabilité en donnant plus de degrés de liberté que leur équivalent en reconfiguration globale, à un coût matériel un peu plus élevé en terme de processeurs, mais beaucoup plus faible en terme de bande passante supplémentaire.

7 Conclusion

Dans cette 6^{ième} et dernière partie de ce mémoire, la question de l'architecture de réseau tolérant aux fautes a été vue sous plusieurs aspects. Cela nous a amenés, d'une part, à déterminer une architecture répondant à des critères liés aux évolutions technologiques, et d'autre part, de nous interroger sur le meilleur schéma de reconfiguration.

Nous nous sommes donc efforcés, tout au long de ce chapitre, de donner les outils pour pouvoir juger de la pertinence des schémas de reconfiguration proposés. Le cas d'un système composé d'ASIC, qui constitue la trame directrice de ce mémoire, a été pris comme référence et les résultats obtenus montrent que le niveau de reconfiguration atteint est tout à fait correct.

La modularité de la technique de reconfiguration permet alors d'augmenter la fiabilité de façon très fine et reste adaptée pour tout type de structure matérielle. Enfin, deux perspectives ont été abordées dans cette partie. La première concerne la possibilité d'utiliser le réseau de reconfiguration pour une reconfiguration fonctionnelle. La seconde utilise les processeurs supplémentaires pour insérer des points froids dans le système et réguler ainsi la température. Pour cette seconde possibilité, un simulateur a été mis au point pour permettre de mesurer l'efficacité des méthodes par rapport à différentes implémentations physiques du réseau. Enfin, une ouverture a été faite en direction de réseaux hétérogènes de type multi-SIMD pour lesquels la reconfiguration locale peut constituer le niveau « bas » de tolérance aux fautes.

Conclusion

Nous avons présenté dans ce mémoire une chaîne complète ayant pour but d'améliorer la fiabilité de calculateurs parallèles SIMD ou multi-SIMD présentant une caractéristique forte d'intégration. Cette chaîne se compose des différents éléments suivants :

1. test de la structure de façon à détecter et localiser les processeurs et interconnexions fautifs ;
2. vérification de la nature non transitoire de la faute ;
3. recouvrement des données correctes de l'élément fautif ;
4. reconfiguration du système si nécessaire, en remplaçant l'élément fautif par un élément "dormant".

Cependant, nous avons plus particulièrement traité les points 1 et 4 de cette chaîne. Ainsi nous proposons :

- un test de fabrication des processeurs du tableau SIMD dans le cas où les processeurs sont réalisés à partir d'une description VHDL d'interconnexions entre blocs, dont la fonction de transfert est connue, ou à partir d'IP logicielle. Ce test de fabrication, basé sur des techniques de test intégré (BIST) est également transparent, c'est à dire qu'il conserve les données précédant le test si aucune faute n'est détectée. Il donne ainsi les moyens de réaliser un test périodique permettant de détecter les fautes permanentes de la structure ;
- un test-en-ligne ou test lors du fonctionnement pour le processeur du tableau SIMD, basé sur des techniques connues d'auto-contrôle par bit de parité. Ce test n'est pas obligatoire selon le type d'applications visé, mais il permet la couverture des fautes transitoires de la structure ;
- un algorithme de reconfiguration permettant de remplacer les processeurs fautifs du réseau par des processeurs supplémentaires en conservant la topologie du réseau. La reconfiguration diffère de celles proposées jusque là, dans le sens où elle associe les trois avantages suivants :
  - ⇒ elle est parfaitement modulaire en nombre de processeurs supplémentaires ;
  - ⇒ elle laisse libre choix de l'architecture de tolérance aux fautes et de son implémentation physique (layout) ;
  - ⇒ elle garantit une faible diminution de performance par une limitation de la longueur des chemins de données.
- une architecture de calculateurs parallèles réaliste (intégrant les contraintes technologiques), tolérante aux fautes, possédant un réseau de type grille ou tore 2-D. A l'architecture proposée, des techniques de reconfiguration locale et globale peuvent être associées. Différents schémas d'amélioration de la fiabilité ont été proposés.

Nous avons évalué sur l'architecture proposée l'amélioration en fiabilité qui pouvait être atteinte. Pour ce faire, nous avons mesuré le taux de reconfiguration obtenu par la méthode de reconfiguration locale, puis nous en avons déduit l'amélioration de la fiabilité R(t) du système complet. Les conclusions de cette étude sont les suivantes : les taux de reconfiguration de la méthode de reconfiguration locale sont très bons et adaptés à notre attente. Ils permettent, selon les différents schémas d'amélioration de fiabilité proposés, d'atteindre des niveaux de fiabilité corrects à élevés pour des coût réduits à moyens.

Enfin, nous avons montré que des schémas à reconfiguration locale sont souvent plus pertinents que les schémas à reconfiguration globale. Par exemple, l'ajout d'un seul processeur supplémentaire par ASIC suffit à améliorer de façon conséquente et souvent suffisante, la fiabilité d'un calculateur formé par quelques dizaines de ces ASIC.

Notre but, qui était de proposer une chaîne complète d'amélioration de fiabilité est donc atteint. Quelques questions restent toutefois en suspens.

Concernant le test, la méthode demande à être améliorée sur deux plans. D'une part, une meilleure justification du taux de couverture, basée sur des fautes réalistes demande à être faite. D'autre part, un test lors du fonctionnement mieux adapté à des structures dont les blocs internes ne sont pas définis structurellement rendrait ce test plus adapté aux méthodes de conception actuelles ou futures. De telles méthodes existent actuellement, leurs pertes en performance sont difficilement acceptables (mis à part pour les systèmes critiques).

Concernant la reconfiguration, une étude plus approfondie sur les effets des fautes dans les interconnexions et les éléments de commutation améliorerait la précision des calculs de fiabilité. Il serait également intéressant, pour ces calculs de fiabilité, de prendre en compte des données physiques, comme le placement des processeurs sur la couche physique et les effets d'une augmentation locale de la température. Le simulateur de Monte Carlo, présenté dans le chapitre 6, peut permettre de réaliser ces études.

Réalisation – Validation

Un ASIC incluant 9 processeurs et le réseau proposé au chapitre 6 a été conçu en technologie MHS 0,6 µm. Chaque processeur, conforme à la description faite au chapitre 3, intègre les ajouts matériels permettant de réaliser le test de fabrication et le test périodique sous forme de BIST transparent. Ainsi, une carte prototype intégrant deux de ces ASIC est actuellement testée au laboratoire du Groupe Architecture Parallèle (GAP) du CEA-LETI. Elle doit permettre de valider l'approche de reconfiguration locale proposée.

Par ailleurs, notre prototype permet de traiter le problème des interconnexions entre ASIC de façon originale. En effet, il doit permettre, à terme, d'utiliser un empilement d'ASIC dont les interconnexions sont réalisées de façon optique par des diodes laser à cavité verticale (VCSELs) [Scheer1997].

Références

Chapitre 1

[CM1991] "The connection machine CM-200 series technical summary," . Cambridge, Massachusetts: Thinking Machine Corporation, 1991.

[CM1992] "CM-5 Technical summary," . Cambridge, Massachusetts: Thinking Machine Corporation, 1992.

[SIA1994] "The national technology roadmap for semiconductors," . San Jose, Calif.: Semiconductor Industry Association, 1994.

[AMSAA1995] AMSAA, "Physics of Failure Newsletter," : AMSAA, 1995.

[ASTE1991] ASTE, "Journée sur la fiabilité des composants électroniques," . Paris, 1991.

[AT&T1997] AT&T, "TR-332-6, Bellcore reliability prediction model," , 1997.

[Batcher1980] K. E. Batcher, "Design of a massively parallel processor," in *IEEE trans. on Comp.*, vol. C-29, n° 9, 1980.

[Bensahel and Bomchil1989] D. Bensahel and G. Bomchil, "Le silicium sur isolant : état de l'art et perspectives d'avenir," in *L'onde électrique*, vol. 69, 1989, pp. 57-66.

[Boppana and Chalasani1995] R. V. Boppana and S. Chalasani, "Fault-tolerant wormhole routing algorithms for mesh networks," in *IEEE trans. on Comp.*, vol. 44, n° 7, 1995.

[Burger and Goodman1997] D. Burger and J. R. Goodman, "Billion-transistor architectures," in *IEEE Computer*, 1997, pp. 46-48.

[Chowdhury and Banerjee1996] A. R. Chowdhury and P. Banerjee, "Algorithm based fault location and recovery for matrix computations on multiprocessor systems," in *IEEE trans. On Comp.*, vol. 45, n° 11, 1996.

[CNET1983] CNET, "Recueil de données de fiabilité du CNET," 1983.

[Collette1993] T. Collette, "Architecture et validation comportementale en VHDL d'un calculateur parallèle dédié à la vision," in *CEA/LETI*: INPG, 1993.

[Collette1998] T. Collette, C. Gamrat, D. Juvin, J. F. Larue, L. Letellieret al., "Symphonie, calculateur massivement parallèle : modélisation et réalisation," in *Traitement du signal*, 1998.

[DoD1995] DoD, "Mil-Hdbk 217-F2 : Reliability Prediction of Electronic Equipment," 1995.

[Duarte1997] R. O. Duarte, M. Nicolaidis, H. Bederr and Y. Zorian, "Fault-secure shifter design : results and implementations," in *ED&TC*, 1997.

[Ducan1990] R. Ducan, “A survey of parallel computer architectures,” in *Computer*, vol. 23, 1990, pp. 5-16.

[Enslow1977] P. H. Enslow, “Multiprocessor organization,” in *ACM Computing Surveys*, vol. 9, 1977.

[Faynot1998] O. Faynot, C. Raynaud and S. L. Palloie, “0.18 um and 0.1 um SOI devices design investigation for 1 V applications,” in *IEEE international SOI 98 Conference*, 1998.

[Flynn1972] M. J. Flynn, “Some computer organizations and their effectiveness,” in *IEEE trans. on Comp.*, vol. C-21, 1972, pp. 948-960.

[Fujiwara and Matsuoka1987] E. Fujiwara and K. Matsuoka, “A self-checking generalized prediction checkers and its use for built-in testing,” in *IEEE trans. on Comp.*, vol. 36, 1987.

[Hayes1988] J. P. Hayes, “Computer architecture and organization,” , 2, Ed.: McGraw-Hill, 1988.

[Juvin1988] D. Juvin, J. L. Basille, H. Essafi and J. Y. Latil, “Sympati2, a 1.5D processor array for image applications,” in *EUSIPCO*. Grenoble, 1988.

[Khakbaz and McCluskey1984] J. Khakbaz and F. J. McCluskey, “Self-testing embedded parity checkers,” in *IEEE trans. on Comp.*, vol. 33, 1984.

[Kim and Shin1996] H. Kim and G. S. Shin, “Design and analysis of an optimal instruction-retry policy for TMR controller computers,” in *IEEE trans. on Comp.*, vol. 45, n° 11, 1996.

[Laprie1985] J. C. Laprie, “Dependable computing and fault tolerance : concepts and terminology,” in *Fault-Tolerant Computing Symposium*, 1985.

[Leighton and Maggs1992] F. T. Leighton and B. M. Maggs, “Fast algorithm for routing around faults in multibutterflies and randomly-wired splitter networks,” in *IEEE trans. On Comp.*, vol. 41, n° 5, 1992.

[Negrini and Stefanelli1986] R. Negrini and R. Stefanelli, “Comparative evaluation of space and time redundancy approaches for WSI processing arrays,” in *Wafer scale integration*, E. s. p. B.V., Ed., 1986, pp. 207-222.

[Nicolaidis1993] M. Nicolaidis, “Efficient implementations of self-checking adders and ALUs,” in *Fault -Tolerant Computing Symposium*, 1993, pp. 586-595.

[Nicolaidis1997] M. Nicolaidis, R. O. Duarte, S. Manich and J. Figueras, “Fault-secure parity prediction arithmetic operators,” in *IEEE Design&Test of Comp.*, 1997, pp. 60-71.

[Noraz1989] S. Noraz, “Application des circuits intégrés autotestables à la sûreté de fonctionnement des systèmes,” in *TIM3*. Grenoble: INPG, 1989.

[Oh1995] C. G. Oh, H. Y. Young and V. K. Raj, “An efficient algorithm-based concurrent error detection for FFT networks,” in *IEEE trans. On Comp.*, vol. 44, n° 9, 1995.

[Palmer1988] J. F. Palmer, “The NCube family of high performance parallel computer systems,” in *Hypercube Concurrent Computers and Applications*, vol. 1, 1988, pp. 847-851.

[Peleg and Weiser1996] A. Peleg and U. Weiser, “MMX technology extension to the Intel architecture,” in *IEEE Micro*, vol. 16, 1996, pp. 42-50.

[Peythieux1998] M. Peythieux, “Centaure : une architecture parallèle hétérogène pour la vision par ordinateur,” in *CEA/LETI*: Université d'Orsay, 1998.

[RD291997] S. R. RD29, “DMILL, a mixed analog-digital radiation hard technology for high energy physics electronics,” : CERN/LHCC, 1997.

[SIA1998] “The national technology roadmap for semiconductors,” . San Jose, Calif.: Semiconductor Industry Association, release 1998

[Sui and Wang1997] P. H. Sui and S. D. Wang, “An improved algorithm for fault-tolerant wormhole routing in meshes,” in *IEEE trans. on Comp.*, vol. 46, n° 9, 1997.

[Tabak1991] D. Tabak, “Les multiprocesseurs,” : Prentice Hall, 1991.

[Tzeng1993] N. F. Tzeng, “Reconfiguration and analysis of a fault tolerant circular butterfly parallel system,” in *IEEE trans. on Parallel and Distributed Systems*, vol. 4, 1993.

[Vaidya1998] N. H. Vaidya, “A case for two-level recovery schemes,” in *IEEE trans. on Comp.*, vol. 47, n° 6, 1998.

[Wakerly1976] J. F. Wakerly, “Microcomputer reliability improvement using triple modular redundancy,” in *IEEE trans. on Comp.*, vol. 64, n° 6, 1976.

Chapitre 2

[JET1994] “Special Issue on Memory Testing,” in *Journal of Electronic Testing, Theory and application*, vol. 5, 1994.

[IEEE1990] I. s. 1149.1, “IEEE standard Test Access Port and boundary-scan architecture,” . Piscataway: JTAG, 1990.

[IEEE1995] I. 1149.5, “IEEE Standard for Module Test and Maintenance Bus (MTM-Bus) Protocol,” . Piscataway: IEEE, 1995.

[Agrawal1993] V. D. Agrawal, C. R. Kime and K. K. Saluja, “A tutorial on Built-In Self Test (part1 : principles),” in *IEEE Design&Test of Comp.*, 1993, pp. 73-82.

[ASTE1991] ASTE, “Journée sur la fiabilité des composants électroniques,” Paris, 1991.

[Ferguson1990] F. J. Ferguson, M. Taylor and T. Larrabee, “Testing for parametric faults in static CMOS circuits,” in *Int’l Test Conference*, 1990, pp. 436-443.

[Flottes and Rouzeyre1996] M. L. Flottes and B. Rouzeyre, “Testability driven synthesis of non-scan data paths,” in *IEEE European Test Workshop*. Montpellier, 1996, pp. 136-142.

[Fritzemeier1989] R. R. Fritzemeier, H. T. Nagle and C. F. Hawkins, “Fundamentals of testability - a tutorial,” in *IEEE trans. on Industrial Electronics*, vol. 36, 1989, pp. 117-128.

[Lee1992a] T. C. Lee, W. H. Wolf and N. K. Jha, “Behavorial synthesis for easy testability in data path scheduling,” in *Int’l Conf. on Computer Design*, 1992, pp. 616-619.

[Lee1992b] T. C. Lee, W. H. Wolf, N. K. Jha and J. M. Acken, “Behavorial synthesis for easy testability in data path allocation,” in *Int’l conf. on Computer Design*, 1992, pp. 29-32.

[Maxwell1991] P. C. Maxwell, R. C. Aitken, V. Johansen and I. Chiang, “The effect of different test sets on quality level prediction : when is 80 % better than 90 % ?,” in *Int’l Test Conf.*, 1991, pp. 358-364.

[Mc Cluskey1985] E. J. Mc Cluskey, “Built-In-Self Test techniques,” in *IEEE Design&Test of Comp.*, 1985, pp. 21-28.

[Nicolaidis1984] M. Nicolaidis, “Conception de circuits intégrés auto-testables pour des hypothèses de pannes analytiques,” : INPG, 1984.

[Papachristou1991] C. A. Papachristou, S. Chiu and H. Harmanani, “A data path synthesis method for self-testable designs,” in *Design Automation Conference*, 1991, pp. 378-384.

[Wakerly1976] J. F. Wakerly, “Microcomputer reliability improvement using triple modular redundancy,” in *IEEE trans. on Comp.*, vol. 64, n° 6, 1976.

[Yang and Peng1998] T. Yang and Z. Peng, “An efficient algorithm to integrate scheduling and allocation in high-level synthesis,” in *Design, Automation and Test in Europe*. Paris, 1998.

Chapitre 3

[JET1994] “Special Issue on Memory Testing,” in *Journal of Electronic Testing, Theory and application*, vol. 5, 1994.

[TIE1989] "Special issue on testing," in *IEEE trans. on Industrial Electronics*, vol. 36, n° 2, May 1989

[Agrawal1993] V. D. Agrawal, C. R. Kime and K. K. Saluja, “A tutorial on Built-In Self Test (part1 : principles),” in *IEEE Design&Test of Comp.*, 1993, pp. 73-82.

[Anderson1971] D.A. Anderson, "Design of self-checking digital networks using coding techniques," *Urbana CSL University of Illinois*, 1971

[Avizienis1997] A. Avizienis, “Toward systematic design of fault-tolerant systems,” in *IEEE Computer*, 1997, pp. 51-58.

[Bennets1994] R.G. Bennets, "Progress in design for test : a personal view," in *IEEE Design&Test of Comp.*, Spring 1994

[Clermidy1998] F. Clermidy, T. Collette and M. Nicolaidis, “On-line periodic test dedicated to parallel computers,” in *Int’l On-Line Testing Workshop*, 1998.

[Duarte1997] R. O. Duarte, M. Nicolaidis, H. Bederr and Y. Zorian, “Fault-secure shifter design : results and implementations,” in *ED&TC*, 1997.

[Dufaza and Zorian1997] C. Dufaza and Y. Zorian, “On the generation of pseudo-deterministic 2-patterns test sequence with LFSRs,” in *ED&TC*, 1997.

[Fujiwara and Matsuoka1987] E. Fujiwara and K. Matsuoka, “A self-checking generalized prediction checkers and its use for built-in testing,” in *IEEE trans. on Comp.*, vol. 36, 1987.

[Gizopoulos1996] D. Gizopoulos, A. Paschalis and Y. Zorian, “An effective BIST Scheme for Datapaths,” in *Int’l Test Conf.*, 1996, pp. 76-85.

[Khakbaz and McCluskey1984] J. Khakbaz and F. J. McCluskey, “Self-testing embedded parity checkers,” in *IEEE trans. on Comp.*, vol. 33, 1984.

[Maxwell1991] P. C. Maxwell, R. C. Aitken, V. Johansen and I. Chiang, “The effect of different test sets on quality level prediction : when is 80 % better than 90 % ?,” in *Int’l Test Conf.*, 1991, pp. 358,364.

[Mc Cluskey1985] E. J. Mc Cluskey, “Built-In-Self Test techniques,” in *IEEE Design&Test of Comp.*, 1985, pp. 21-28.

[Mukherjee1995] N. Mukherjee, M. Kassab, J. Rasjski and J. Tyszer, “Arithmetic Built-In Self Test for high-level synthesis,” in *VLSI test symposium*, 1995, pp. 132-139.

[Mukherjee1997] N. Mukherjee, J. Rajski and J. Tyszer, “Design of Testable Multipliers for Fixed-Width Data Paths,” in *IEEE trans. On Comp.*, vol. 46, n° 7, 1997.

[Nicolaidis1992] M. Nicolaidis, “Transparent BIST for RAMs,” in *International Test Conference*. Baltimore, 1992.

[Nicolaidis1993] M. Nicolaidis, “Efficient implementations of self-checking adders and ALUs,” in *Fault -Tolerant Computing Symposium*, 1993, pp. 586-595.

[Nicolaidis1996] M. Nicolaidis, “Theory of transparent BIST for RAMs,” in *IEEE trans. On Comp.*, vol. 45, n° 10, 1996.

[Nicolaidis1997] M. Nicolaidis, R. O. Duarte, S. Manich and J. Figueras, “Fault-secure parity prediction arithmetic operators,” in *IEEE Design&Test of Comp.*, 1997, pp. 60-71.

[Nikolos1989] D. Nikolos, “Design of self-testing embedded parity checkers using two-input XOR gates,” in *Int’l Conf. Fault-Tolerant Systems and Diagnostics*, 1989, pp. 158-162.

[Nikolos1998] D. Nikolos, “Optimal self-testing embedded parity checkers,” in *IEEE trans. on Comp.*, vol. 47, n° 3, 1998.

[Rajski and Tyszer1993] J. Rajski and J. Tyszer, “Test Responses Compaction in Accumulators with Rotate Carry Adders,” in *IEEE trans. on CAD*, vol. 12, 1993.

[Smith1980] J.E. Smith and G. Metze, "Strongly fault-secure logic networks," in *IEEE trans. on Comp.*, Vol C-29, n° 6, pp 442-451, 1980

[Tarnik1995] S. Tarnik, “Embedded parity and two-rail TSC checkers with error-memorizing capability,” in *Int’l On-Line Test Workshop*, 1995, pp. 221-225.

[Van de Goor and Tlili1998] J. Van de Goor and I. B. S. Tlili, “March tests for word-oriented memories,” in *Design, Automation and Test in Europe*. Paris, 1998.

Chapitre 4

[Bae and Bose1996] M. Bae and B. Bose, “Spare processor allocation for fault tolerance in torus-based multicomputers,” in *26th Fault Tolerant Computing Symposium*, 1996.

[Batcher1980] K. E. Batcher, “Design of a massively parallel processor,” in *IEEE trans. On Comp.*, vol. 29, n° 9, 1980.

[Belkhale and Banerjee1992] K. P. Belkhale and P. Banerjee, “Reconfiguration strategies for VLSI processor arrays and trees using a modified diogenes approach,” in *IEEE trans. On Comp.*, vol. 41, n° 1, 1992.

[Bruck1993] J. Bruck, R. Cypher and C. T. Ho, “Fault tolerant meshes and hypercubes with minimal numbers of spares,” in *IEEE trans. On Comp.*, vol. 42, n° 9, 1993.

[Bruck1992] J. Bruck, R. Cypher and D. Soroker, “Tolerating faults in hypercubes using subcube partitioning,” in *IEEE trans. on Comp.*, vol. 41, n° 5, 1992.

[Chean and Fortes1989] M. Chean and J. A. B. Fortes, “FUSS : a reconfigurable scheme for fault tolerant processor arrays,” in *International Workshop hardware fault tolerance in multiprocessors*, 1989.

[Chean and Fortes1990] M. Chean and J. A. B. Fortes, “A taxonomy of reconfiguration techniques for fault-tolerant processor arrays,” in *IEEE Computer*, 1990, pp. 55-69.

[Chen1997] Y. Y. Chen, S. J. Upadhyaya and C. H. Cheng, “A comprehensive reconfiguration scheme for fault-tolerant VLSI/WSI array processors,” in *IEEE trans. On Comp.*, vol. 46, n° 12, 1997.

[Dutt and Mahapatra1997] S. Dutt and N. R. Mahapatra, “Node-covering, error-correcting codes and multiprocessors with very high average fault tolerance,” in *IEEE trans. on Comp.*, vol. 46, n° 9, 1997.

[Jean1990] J. S. N. Jean, H. C. Fu and S. Y. Kung, “Yield enhancement for WSI Array Processors using two and a half Track switches,” in *International Conference On Wafer Scale Integration*. San Francisco, 1990, pp. 243-250.

[Jean and Kung1989] S. N. Jean and S. Y. Kung, “Necessary and sufficient conditions for reconfigurability in single-track switch WSI arrays,” in *Int’l Conf. on Wafer Scale Integration*, 1989, pp. 401-412.

[Koren1981] I. Koren, “A reconfigurable and fault tolerant VLSI multiprocessor,” in *8th Symp. Computer architecture*. Los Alamitos, Calif., 1981, pp. 425-442.

[Kung1989] S. Y. Kung, S. N. Jean and C. W. Chang, “Fault tolerant array processors using single track switches,” in *IEEE Trans. On Comp.*, vol. 38, n° 4, 1989.

[Kuo and Fuchs1987] S. Y. Kuo and W. K. Fuchs, “Efficient spare allocation for reconfigurable arrays,” in *IEEE design and test of Comp.*, vol. 4, n° 1, 1987.

[Leighton and Leiserson1986] F. T. Leighton and C. Leiserson, “A survey of algorithms for integrated wafer scale systolic arrays,” : MIT/LCS, 1986.

[Li and Stout1991] H. Li and Q. F. Stout, “Reconfigurable massively parallel computers,” , H. Li and Q. F. Stout, Eds.: Prentice Hall, 1991.

[Lombardi1989] F. Lombardi, M. G. Sami and R. Stefanelli, “Reconfiguration of VLSI arrays by covering,” in *IEEE Trans. On CAD of IC and Systems*, vol. 8, n° 9, 1989.

[Mahapatra and Dutt1996] N. R. Mahapatra and S. Dutt, “Hardware-efficient and highly-reconfigurable 4- and 2- Track fault tolerant designs for mesh-connected multicomputers,” in *International Symposium On Fault Tolerant Computing*, 1996, pp. 272-281.

[Moore and Mahat1985] W. R. Moore and R. Mahat, “Fault-tolerant communications for WSI of a processor array,” in *Microelectronics and reliability*, vol. 25, n° 2, 1985.

[Negrini1986] R. Negrini, M. Sami and R. Stefanelli, “Fault tolerance techniques for array structures used in supercomputing,” in *Computer*, vol. 19, n° 2, 1986.

[Raghavendra1992] C. S. Raghavendra, P. J. Yang and S. B. Tien, “Free dimensions - an effective approach to achieving fault tolerance in hypercubes,” in *Fault Tolerant Computing Symposium*, 1992, pp. 170-177.

[Raghavendra1995] C. S. Raghavendra, P. J. Yang and S. B. Tien, “Free dimensions - an effective approach to achieving fault tolerance in hypercubes,” in *IEEE trans. on Comp.*, vol. 44, n° 9, 1995.

[Rosenberg1983] A. L. Rosenberg, “The diogenes approach to testable fault-tolerant arrays of processors,” in *IEEE trans. On Comp.*, vol. C32, n° 10, 1983.

[Roychowdhury1990] P. Roychowdhury, J. Bruck and T. Kailath, “Efficient algorithms for reconfiguration in VLSI/WSI arrays,” in *IEEE Trans. On Comp.*, vol. 39, n° 4, 1990.

[Singh1988] A. D. Singh, “Interstitial redundancy : An area efficient fault-tolerant scheme for large area VLSI processor arrays,” in *IEEE trans. on Comp.*, vol. C-37, 1988.

[Varvarigou1991] T. A. Varvarigou, V. P. Roychowdhury and T. Kailath, “Reconfiguring processor arrays using multiple-tracks models,” in *International Conference on Wafer Scale Integration*, 1991, pp. 307-311.

[Varvarigou1993] T. A. Varvarigou, V. P. Roychowdhury and T. Kailath, “Reconfiguring processor arrays using multiple-track models: the 3-track-1-spare approach,” in *IEEE trans. on Comp.*, vol. 42, n° 11, 1993.

Chapitre 6 et conclusion

[Boku1997] T. Boku, H. Nakamura, K. Nakazawa and Y. Iwasaki, “The architecture of massively parallel processor CP-PACS,” 1997.

[Lombardi1989] F. Lombardi, M. G. Sami and R. Stefanelli, “Reconfiguration of VLSI arrays by covering,” in *IEEE Trans. On CAD of IC and Systems*, vol. 8, n° 9, 1989.

[Scheer1997] P. Scheer, T. Collette and P. Churoux, “Massively parallel computers using optical interconnects - the synoptique project,” in *Optical interconnections and parallel processing : trends at the interface*, P. Berthome and A. Ferreira, Eds.: Kluwer Academic Publishers, 1997.

Publications et dépôts de brevets

F. Clermidy, T. Collette and M. Nicolaidis, “On-line periodic test dedicated to parallel computers,” in *Int’l On-Line Testing Workshop (IOLTW)*, July 1998.

F. Clermidy, T. Collette and M. Nicolaidis, “Periodic Test and Structural Fault-Tolerance : a Strategy to Attain High Reliability in Embedded Parallel Computers” in *Int’l On-Line Testing Workshop (IOLTW)*, July 1999.

F. Clermidy, T. Collette, M. Nicolaidis, "SYNTOL : SYstème parallèle Naturellement TOLérant aux fautes", *atelier architectures embarquées de traitement du signal du CNES*, novembre 1999

F.Clermidy, T. Collette, M. Nicolaidis, " A new placement algorithm dedicated to parallel computers : bases and application" *In Pacific Rim International Symposium on Dependable Computing (PRDC),* December 1999

F. Clermidy, F. Blanc, T. Collette, M. Nicolaidis, "SYNTOL : SYstème parallèle Naturellement TOLérant aux fautes", in *Workshop in Algorithm Architecture Adequation (AAA)*, Janvier 2000

F. Clermidy, T. Collette, "réseau de processeurs parallèles avec tolérance aux fautes de ces processeurs, et procédé de reconfiguration applicable à un tel réseau", demande de brevet no 99.08553, juillet 99

F. Clermidy, T. Collette, "procédé de reconfiguration applicable à un réseau d’éléments fonctionnels identiques", demande de brevet no 99.08554, juillet 99

Annexe

1 La sûreté de fonctionnement

1.1 Introduction

Pour des raisons de clarté, les termes anglais seront indiqués pour chaque terme spécialisé. En effet, le vocabulaire employé en sûreté de fonctionnement est très précis, et chaque mot à une définition bien spécifique. Une classification précise des termes concernant la sûreté de fonctionnement a été faite par JC Laprie [Laprie1985]. Cette classification est reprise dans la suite afin de définir précisément le domaine. Le terme-titre de cette section est tout d'abord défini :

Sûreté de fonctionnement d'un système (dependability) : qualité d'un service délivré par le système tel qu'une confiance peut justifiablement être placée dans ce service (le service étant le comportement du système perçu par l'extérieur).

Concernant la sûreté de fonctionnement, on verra successivement :

- les défauts du système qui mettent en cause la sûreté de fonctionnement: fautes (faults), erreurs (errors) et défaillances (failures).
- les moyens pour atteindre la sûreté de fonctionnement : éviter la faute (fault-avoidance), faire de la tolérance aux fautes (fault-tolerance), faire de la suppression d'erreur (error-removal) et faire de la prévision d'erreur (error-forecasting).
- les mesures de la sûreté de fonctionnement qui sont la fiabilité (reliability) et la disponibilité (availability) du système.

1.2 Les défauts du système

- Une défaillance (*failure*) est une déviation du service délivré par le système par rapport au service spécifié.
- Une erreur (*error*) est un défaut de l'état du système qui peut mener à une défaillance.
- Une faute (*fault*) est la cause d'une possibilité d'erreur. Une faute ne conduisant pas à une erreur est appelée latente.

D'un autre point de vue, on peut dire que l'erreur est une manifestation d'une faute dans le système et que la défaillance est une manifestation d'une erreur dans le service. Le but de la sûreté de fonctionnement est donc d'éviter qu'une défaillance survienne, par exemple en traitant la faute dès qu'elle apparaît, ou de minimiser ces temps de défaillance par rapport aux temps de fonctionnement correct.

Nous ne parlerons ici que de l'amélioration de la fiabilité des systèmes par rapport aux fautes physiques matérielles qui peuvent survenir. D'autres fautes, qui sont les fautes de conception et les fautes logicielles ne sont pas traitées ici.

1.3 Les méthodes afin d'atteindre le but de sûreté de fonctionnement

Les moyens pour améliorer la sûreté de fonctionnement peuvent se classer dans deux catégories :

- Les méthodes pour éviter les fautes (*fault avoidance*) qui répondent à la question "comment prévenir *par construction* l'apparition de fautes?".
- Les méthodes pour tolérer les fautes (*fault tolerance*) qui répondent à la question "comment fournir *par redondance* un service conforme aux spécifications en dépit des fautes pouvant apparaître ?".

Mais il faut également prévoir des méthodes de validation de la sûreté de fonctionnement que l'on peut mettre dans deux autres classes :

- Suppression d'erreur (*error-removal*), méthode qui répond à la question "comment minimiser, *par vérification*, la présence d'erreurs latentes?".
- Prévision d'erreur (*error-forecasting*) qui répond à la question "comment estimer, *par évaluation*, la présence, la création et la conséquence d'erreurs?".

1.4 La mesure de la sûreté de fonctionnement

La vie d'un système est perçue, du point de vue utilisateur, comme la succession de deux états du service délivré, le premier correspondant au service délivré comme spécifié (*accomplishment*) et le deuxième correspondant au service qui diffère de la spécification (*interruption*).

Les transitions entre ces deux états sont la défaillance (*failure*) et la restauration (*restoration*).

Pour quantifier cela, on utilise deux mesures:

- La fiabilité (*reliability*) mesure le temps moyen de la continuité du service, c'est à dire le temps compris entre la mise en route du système et l'apparition d'une défaillance de ce système par rapport au service demandé.
- La disponibilité (*availability*) mesure la qualité du service en tenant compte des temps d'interruption, c'est donc la mesure du temps de bon fonctionnement par rapport au temps de défaillance.

Résumé :

La sûreté de fonctionnement des systèmes électroniques est un sujet de plus en plus complexe en raison des avancées technologiques et architecturales. Les structures comportant à la fois un grand nombre de composants et conçues à partir de technologies agressives sont parmi celles dont les problèmes de fiabilité doivent être considérés avec la plus grande attention. Parmi ces structures, les calculateurs parallèles intégrés, puissants accélérateurs de calcul dans un volume réduit, se doivent d'assurer un niveau de fiabilité élevé à ses utilisateurs. Dans cette thèse, nous proposons une méthode d'amélioration de la fiabilité dédiée à ces calculateurs fondée sur des techniques originales de test et de tolérance aux fautes. La méthode de tolérance aux fautes consiste en une reconfiguration du réseau sur 2 niveaux de hiérarchie physique, fondée sur la connaissance permanente de l'état de la structure obtenue par un test périodique ou concurrent. Nous montrons alors comment il est possible, au moyen d'un ajout matériel minimisé et modulaire, d'atteindre des taux de fiabilité équivalents à ceux d'un des composants de la structure d'origine.

Abstract :

With architectural and technologies advances, the way to assure electronic structures dependability becomes more and more complex. This problem is particularly crucial for structures composed of a large number of components and realized with aggressive technologies. Integrated parallel computers are such structures. They used to give high processing rate in a low volume, with a high confidence that can be put in the result. In this thesis, we propose a scheme to improve the reliability of parallel computers, based on original test and fault-tolerance schemes. In our case, the test scheme uses periodic and concurrent methods in order to permanently know the structure state. If some faulty component is found, the fault-tolerance scheme is then performed by two physical levels of reconfiguration of the structure network. We show, in the conclusion of the thesis, that with minimal and scalable hardware overhead, reliability comparable to the one of a component of the non fault-tolerant structure can be achieved.

ISBN 2-913329-41-1 (broché)

ISBN 2-913329-42-X (format électronique)