Université de Montréal

Méthode de Simulation Aléatoire Guidée par un Algorithme Génétique pour la Vérification du Design de Circuits Numériques

par

Pierre Faye

Département d'informatique et de recherche opérationnelle

Faculté des arts et des sciences

Mémoire présenté à la faculté des études supérieures
en vue de l'obtention du grade de
Maître ès sciences (M.Sc.)
en Informatique

octobre, 1999

Université de Montréal
Faculté des études supérieures

Ce mémoire intitulé:

Méthode de Simulation Aléatoire Guidée par un Algorithme Génétique pour la Vérification du Design de Circuits Numériques

présenté par

Pierre Faye

a été évalué par un jury composé des personnes suivantes:

El Mostapha Aboulhamid, président-rapporteur

Eduard Cerny, directeur

Pierre L'Écuyer, membre du jury.

Mémoire accepté le:

SOMMAIRE

Nous décrivons une méthode de vérification de designs de circuits numériques par simulation aléatoire dirigée. L'objectif de la méthode est l'amélioration de la qualité de la vérification en utilisant des séquences de signaux d'entrées aléatoires biaisés par l'information régulièrement reçue en cours d'expérience sur les variations de l'étendue de la couverture de design atteinte par la simulation. L'algorithme qui contrôle la simulation est inspiré par des algorithmes d'optimisation génétique. Les critères d'optimisation sont les comptes d'activations de divers vérificateurs de groupes de propriétés, chacun étant rattaché à une fonctionnalité spécifique du design.

L'information génétique (le chromosome) consiste en un vecteur de valeurs qui paramétrisent les distributions de nombres aléatoires liées aux entrées libres du système simulé, constitué du design sous vérification et d'un modèle non déterministe de son environnement. Les entrées libres, contrôlées par des générateurs de nombres aléatoires munis de leur distribution, sont responsables des choix non déterministes du système.

Des résultats expérimentaux de l'application de cette méthode sur un design RTL Verilog constitué d'environ 6000 portes logiques équivalentes montrent qu'il est possible d'atteindre le même niveau de couverture de design obtenu en utilisant la méthode traditionnelle de simulation aléatoire pure mais en réduisant de moitié le nombre de vecteurs d'entrées à appliquer au circuit. Autrement dit, avec le même

nombre de vecteurs d'entrées, les comptes d'activations des propriétés sont doublés. Des résultats similaires ont été obtenus avec un design constitué de 800 K portes logiques équivalentes.

mots clés: vérification de designs, circuits numériques, micro-électronique, simulation, algorithmes génétiques.

TABLE DES MATIÈRES

LISTE DES TABLEAUX

LISTES DES FIGURES

REMERCIEMENTS

A ma mère, la femme de ma vie,
pour son amour si chaleureux,
son soutien si précieux et...
son sourire si merveilleux.

A mon défunt père, pour son omniprésence
dans toutes mes activités.

J'espère que ces quelques lignes sauront exprimer à quel point je suis fier et pleinement satisfait d'avoir été remarquablement bien encadré dans mes recherches par le Professeur Eduard Cerny pendant toute la durée de cette étude. A plusieurs reprises, j'ai eu l'occasion d'admirer le côté humain, quasi paternel d'Eduard qui est, et qui restera pour moi au delà d'un professeur expérimenté, un modèle de conduite.

Je tiens à remercier la compagnie Nortel Networks pour son aide financière, et en particulier Alan Silburt, pour nous avoir permis de tester notre méthodologie.

J'exprime une profonde gratitude envers tous les membres de ma famille et en particulier: mes soeurs Marie Odile, Marie Brigitte et Marie Christine, mon petit frère Yves Marie, mes beaux frères Alioune et Karim, ma belle soeur Yéta, mes cousins Christian, Henri Michel, Edouard, et... *"last but not least"*, Nicou, pour leur soutien moral permanent malgré la distance qui nous sépare. J'adresse une pensée particulière à l'égard de mon oncle Julien Jouga qui nous a toujours soutenus, et en particulier dans les moments les plus difficiles.

Enfin je voudrais remercier tout particulièrement mes amis Saidou Kane, Cheikh Sarr, Joséphine Pelleu, Mohamed Zaakoun, Moctar N'diaye et Mohamed N'diaye qui m'ont été d'un support fraternel nécessaire pour la réalisation de ce travail.

CHAPITRE PREMIER

Introduction

L'industrie du semiconducteur a connu une évolution des plus spectaculaires depuis les premiers circuits intégrés du début des années 1970s. Un microprocesseur typique tel que le *8080* d'Intel était alors constitué d'environ 5000 transistors. En 1999, le *Pentium Pro* d'Intel en compte plus de 5 millions. Cette maturité phénoménale a entraîné une complexité considérable dans le design et la fabrication des circuits micro-électroniques qui s'est inévitablement répercutée au sein des laboratoires de recherche spécialisés en vérification de design de matériel informatique.

La vérification de matériel micro électronique et informatique est essentielle, voire indispensable, surtout lorsque l'on sait que plusieurs cas d'erreurs ont été découverts trop tard dans le processus de construction, et même parfois après la production commerciale sur le marché. A titre d'exemple, on peux citer le dernier né (novembre 1997) de la famille des "*bug*" dans les processeurs Pentium d'Intel, qui se fait appeler: "Pentium FO bug". Non seulement, ce *bug* gèle l'activité interne des ordinateurs à processeur Pentium MMX qui sont utilisés par des centaines de millions d'usagers à travers le monde, mais l'hypothèse selon laquelle d'autres processeurs de

la même famille (comme le Pentium Pro) pourraient aussi être atteints par l'anomalie n'est pas écartée.

Un procédé relativement récent est l'utilisation de techniques de vérification formelle comme outil de garantie de l'exactitude (*correctness*) du matériel. La vérification formelle est dans un certain sens, une preuve mathématique. Si un théorème mathématique est vrai, alors il en est de même pour toutes les valeurs particulières sur lesquelles il s'applique. De façon similaire, si le design d'un circuit est formellement vérifié, alors l'exactitude de ce matériel s'applique pour toutes ses entrées possibles. La vérification de tous les états possibles du circuit est donc implicitement effectuée dans une méthodologie de vérification formelle. Une certaine polarisation s'est créée dans la communauté des chercheurs en vérification formelle: Ceux qui soutiennent les méthodes automatisées basées sur les machines à états finis (et en particulier "*model-checking*") d'un côté et les adeptes de la méthode de "*theorem-proving*" de l'autre. Il convient néanmoins de noter que la fusion des 2 méthodes est actuellement utilisée. Ce mémoire n'étant pas axé sur l'application des méthodes formelles à la vérification de design, nous en présenterons un très bref survol en mettant surtout l'accent sur les inconvénients majeurs de telles méthodes. De plus amples informations et références pourront être obtenues dans [1, 2].

Theorem-proving est une méthode déductive de par sa nature. L'automatisation est possible jusqu'à un certain point. Cependant, la plupart des "*theorem-provers*" disponibles aujourd'hui sont au mieux semi-automatisés dans le sens où ils ont besoin d'une certaine interaction humaine pour guider le processus de recherche de preuve.

D'autres part, les systèmes de theorem-proving sont très généraux d'un point de vue applicatif. En effet, la logique permet la représentation virtuelle de tout le bagage mathématique nécessaire à la preuve de théorème dans un environnement spécifique. Elle est donc aussi bien capable de formaliser d'importants systèmes matériels sophistiqués que de simples propriétés. Malheureusement, c'est justement cette grande généralité qui lui fait payer le prix d'une importante complexité qui croit exponentiellement avec la taille des circuits à vérifier.

A l'opposé, "*model-checking*" est une activité relativement plus simple. Dans la plupart des cas, un algorithme clair peut être fourni, et ceci de façon complètement automatisée. Malheureusement, la méthode présente deux inconvénients majeurs. D'un part, une telle méthode n'est pas aussi générale que celle de "*theorem-proving*". Un système de vérification ne marchera que pour le type de logique et de modèles pour lesquels il a été conçu. D'autre part, la méthode présente certaines faiblesses dans sa capacité à intégrer la hiérarchisation dans ses systèmes. La plupart des approches "model-checking" utilisent des descriptions de matériel non hiérarchiques, basées sur une machine à états finis. Dans cette situation, une augmentation de composants matériel résulte en une explosion du nombre global d'états (*state explosion problem*), se traduisant par une baisse des performances (voire une impossibilité d'application) sur les problèmes de grande taille. L'introduction de techniques comme "symbolic model checking" ou "modular verification" tente de réduire le problème mais ne le résolvent pas.

Parmi les techniques et outils que l'on retrouve dans la littérature existante (certains ayant d'ailleurs déjà été commercialisés), on peux citer: l'amélioration de la productivité de la vérification par de meilleurs environnements de construction de bancs d'essais (testbenches) [3, 4]. La réduction du nombre de cas de bancs d'essais en utilisant des séquences d'entrées aléatoires [3, 4]. L'amélioration de l'observabilité par des vérificateurs de propriétés générés automatiquement [5]; L'amélioration de l'évaluation de la couverture de code par des outils logiciels [6] ou basée sur l'information provenant des états / transitions du réseau de machines à états finis sous-jacent au design [5]; L'utilisation de la vérification de modèle (Symbolic Model Checking) pour la vérification de propriétés spécifiées en logique temporelle [7, 8] ou utilisant des tests d'inclusions de langages d'automate w [9]. L'exploration partielle de l'espace d'états est aussi en développement [10]. Les preuves formelles (theorem proving) à un niveau de design abstrait ont aussi été utilisées [9, 11].

Les méthodes et techniques formelles se révélant être peu efficaces lorsqu'il s'agit de vérifier des designs RTL de grandes tailles, qu'en est il de la simulation? La quantité de lignes de code utilisée pour écrire des modules de test croit beaucoup plus rapidement que la taille des design à tester. La taille est déterminée par deux facteurs: le nombre de différentes caractéristiques (tirées des spécifications du système) à vérifier, et la complexité des séquences d'entrées qui doivent explorer les différents aspects du modèle RTL. Le premier facteur peut être amélioré en utilisant une équipe indépendante pour développer un modèle comportemental du système basé sur ses spécifications. Le modèle sert alors de référence dans la comparaison avec les sorties

du modèle RTL. Une différence détectée sera analysée pour déterminer quel modèle est erroné. Il reste à résoudre le problème du développement des séquences d'entrées appropriées.

Une approche possible qui tend à réduire la taille et le nombre des modules de test est d'utiliser de courtes séquences déterministes pour éliminer d'éventuelles erreurs évidentes de design, et alors d'utiliser des séquences longues aléatoirement générées pour atteindre les autres parties de l'espace d'états d'une façon moins dirigée. Ici, à la place de temps et de génie humain, le temps du CPU est utilisé dans de longues séquences de simulation pour atteindre une couverture de design suffisamment importante. L'inconvénient de cette approche est qu'elle ne garantit pas, malgré la longueur des séquences aléatoires (temps de simulation élevé), que les cas isolés (*corner cases*) seront atteints. Cet aspect dépend beaucoup plus des distributions aléatoires et de la complexité du design. Il convient alors de se demander s'il est possible de guider la simulation aléatoire de façon à atteindre une meilleure couverture sans éliminer son caractère aléatoire. Evidemment, l'information sur la structure du design pourrait aider à sélectionner des distributions aléatoires plus propices à une large couverture, mais s'il s'avère que les résultats obtenus ne soient pas aussi intéressants que ceux escomptés, il n'est pas toujours évident de réajuster les paramètres de façon relativement simple.

Nous nous sommes proposés alors de répondre à cette question en imaginant une façon dynamique de mesurer la couverture atteinte à intervalles de temps réguliers

pendant le déroulement de la simulation afin d'utiliser cette information phare pour réajuster régulièrement les paramètres des distributions aléatoires qui contrôlent la simulation, avec pour objectif d'explorer l'espace des séquences d'entrées au circuit le mieux possible. Le dynamisme de cette méthode est rendu possible à l'aide de l'utilisation d'une sous classe particulière de l'ensemble des algorithmes évolutionnaires: les Algorithmes Génétiques (AGs).

Notre méthode de vérification peut se résumer de la façon suivante: On exécute une population de simulations relativement courtes, chacune étant conduite par un vecteur dont les coordonnées paramétrisent les distributions de probabilité des générateurs aléatoires liés aux entrées primaires du circuit. Durant chacune des simulations, la couverture est mesurée par des observateurs (*checkers*) qui analysent les signaux internes ou échangés entre le design et son environnement. Le résultat de cette évaluation affecte un rang qualitatif au vecteur conducteur correspondant. Les meilleurs (plus haut rang) vecteurs sont alors sélectionnés, combinés entre eux et mutés, pour former une nouvelle population de vecteurs. Ce processus est continuellement réitérer jusqu'à ce que le niveau de couverture relevé atteigne une proportion satisfaisante.

Nous avons expérimenté notre méthode sur des designs, écrits en langage HDL Verilog, inspirés de deux applications industrielles. La première est un sous système de réordonancement de transactions constitué d'environ 6000 portes logiques

équivalentes tandis que la deuxième est un circuit de multiplexage de paquets constitué d'environ 800,000 portes logiques équivalentes.

Nous disposions au début de ce projet des Designs du circuit issus de la première application industrielle et de son environnement, codés en langage HDL Verilog, ainsi que d'un ensemble de propriétés qui devaient être vérifiées par le design.

Nos principales contributions ont été:

- L'application des algorithmes dits évolutionnaires sur un problème bien particulier de vérification de design de circuits électroniques.
- L'utilisation originale des paramètres de distributions de nombres aléatoires utilisées par la simulation comme information de base (chromosome) de l'algorithme génétique.
- L'estimation de la couverture de vérification atteinte par le nombre d'activations de différentes propriétés à vérifier par le système, liées aux différentes fonctionnalités du système.

Ce mémoire est organisé comme suit: nous présenterons au deuxième chapitre une étude sur les algorithmes génétiques. Le troisième chapitre est consacré à la description de la méthodologie sous-jacente à notre expérience, l'organisation du

système et l'environnement de vérification. Au chapitre quatre, nous exposerons et interpréterons les résultats expérimentaux obtenus. Enfin une conclusion sur le travail réalisé et des propositions sur de plus amples investigations directement reliées à ce sujet sont présentées au chapitre cinq.

CHAPITRE DEUX

Algorithmes Génétiques

"Individuals from a population are bred together to produce new individuals. Those that perform badly die off, and those that perform well are bred again to produce even better individuals"

Charles Darwin.

2.1 Introduction

Charles Darwin nous fait remarquer dans [12] que la sélection artificielle des espèces est une pratique très ancienne, remontant aux premiers agriculteurs éleveurs qui ont très vite compris que pour obtenir un cheptel (ou une récolte) de qualité supérieure, il fallait savoir repérer et sélectionner les espèces dotées de caractéristiques intéressantes et les laisser se reproduire entre elles. Aujourd'hui, le processus de sélection artificielle est largement répandu aussi bien dans l'agriculture et l'élevage industriel que dans les laboratoires de recherche génétique. Des exemples comme l'obtention de troupeaux d'animaux dociles (mouflons), à partir de moutons sauvages ou encore le clonage d'un mammifère adulte donnant vie à une brebis (Dolly), montrent bien les progrès gigantesques de la génétique. Il devient alors très tentant

d'envisager l'utilisation de méthodes informatiques inspirées de cette mécanique génétique.

2.2 Un bref historique de la génétique

La génétique est une science théorisée voilà 130 ans, mais dont les applications se perdent dans la nuit des temps. Cette section débutera par un bref résumé chronologique du génie génétique avant de mettre le doigt sur le lien de plus en plus apparent entre la science de l'information et la génétique.

- 1865. Le moine tchèque Gregor Mendel formule les premières lois de la génétique incluant la théorie des chromosomes et des gènes.

- 1869. Découverte d'une substance appelée nucléine dans le laboratoire de l'Université de Tubingen, en Allemagne par le médecin suisse Friedrich Mischer.

- 1926. Deux acides nucléiques: l'ADN (pour acide désoxyribonucléique) et l'ARN (acide ribonucléique) sont finalement identifiés après les efforts de savants réputés tels que Albrecht Kossel (Allemagne) et Phoebus Levene (E.U.). On ignore encore complètement le lien entre ces deux substances et les gènes découverts par Mendel.

- 1944. Oswald T. Avery, Colin MacLeod et Maclyn McCarthy du Rockefeller Institute for Medical Research à New York publient un article qui établit que l'ADN est bien la molécule qui supporte les gènes.

- 1950. La course à la découverte de la structure de l'ADN est alors pleinement engagée. James Watson et Francis Crick, publient alors un article intitulé *A structure for Deoxyribose Nucleic Acid* ([41]) où la populaire forme de double hélice est introduite. Cette dernière découverte permet d'établir que les quatre bases, l'Adénine, la Cytosine, la Thymine et la Guanine forment l'alphabet fondamental à partir duquel sont codés les génomes de tous les organismes vivants. La reproduction des gènes et des protéines se nomme réplication. Intervenir sur l'ordre des bases permet de modifier les caractères physico-chimiques d'un individu, tels que la couleur de ses cheveux ou la présence du gène d'une maladie héréditaire comme l'hémophilie.

- La table était mise pour les manipulations génétiques. L'art de modifier le génome des organismes vivants a connu ses premiers balbutiements en 1973, quand des gènes étrangers ont été introduits dans une bactérie par des chercheurs américains. Vingt trois ans plus tard, le 5 juillet 1996, naissait le premier clone réussit d'un mammifère adulte (c'est-à-dire qu'elle est la copie exacte d'un autre mouton), une belle brebis appelée Dolly (Figure 1).

Figure 1: Hello Dolly !

2.3 Le lien avec l'informatique

L'ADN nous apparaît dés lors comme un support d'information. Il devient donc possible, comme nous le fait remarquer Renaud Dumeur [14], de considérer la mécanique génétique entrant en jeu dans l'évolution des espèces comme un système de traitement de l'information, un programme. Certes, ce programme est exécuté par une machine ultra-rapide et massivement parallèle: l'univers et les lois physiques qui régissent les interactions entre atomes. Toutefois, en utilisant un modèle d'ADN extrêmement simplifié et en y appliquant un faible nombre d'opérations, il est possible de créer des Algorithmes Génétiques [15] qui permettent la recherche de solutions à des problèmes posés à l'homme, tout en exhibant des propriétés similaires à celles que l'on reconnaît au mécanisme de l'évolution naturelle: la robustesse et l'aptitude à explorer une vaste gamme de solution potentielles.

Les algorithmes génétiques (AGs) sont des méta-heuristiques. Une heuristique est une méthode permettant de trouver en un temps de calcul raisonnable, une solution satisfaisante à un problème compliqué. Cette solution ne sera pas la meilleure (ce serait trop long à calculer) mais elle s'en rapprochera parfois beaucoup. Un algorithme génétique est qualifié de "méta-heuristique" car il s'agit d'une méthode générale à spécialiser pour chaque problème. En tant que méthode de recherche, l'approche génétique consiste à utiliser un ensemble de solutions potentielles auxquelles on attribue des valeurs (voir figure 2).

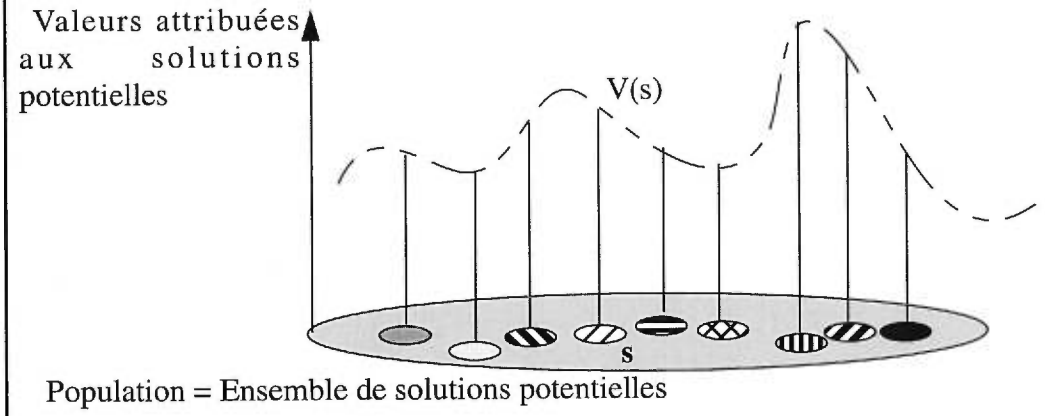

Figure 2: Population de solutions d'un GA

En assimilant une solution à un individu et les caractéristiques de la solution au patrimoine génétique de l'individu (les composants du génome), le GA fait évoluer une population d'individus en y propageant les caractéristiques des individus les plus valables qui, suivant la métaphore évolutionniste, sont les plus adaptés. Les individus ainsi soumis à une sélection et une évolution vont s'adapter et trouver une niche écologique qui correspond à un maximum dans l'espace de recherche. Les maximums locaux peuvent en outre être surmontés par le système car les individus qui la composent sont, tout comme dans la nature, soumis à des mutations aléatoires engendrant de nouvelles caractéristiques.

2.4 Terminologie propre aux algorithmes génétiques

Avant d'aller plus en profondeur dans la description d'un GA, introduisons le vocabulaire que nous allons utiliser:

- Gène: Une caractéristique élémentaire d'une solution au problème que le GA traite.
- Individu: Dans le cadre d'un GA traditionnel, l'individu est réduit à un chromosome, c'est à dire un ensemble de caractéristiques élémentaires (gènes).
- Chromosome: Un GA classique traite des individus constitués d'une seule séquence de gènes, un chromosome, alors que les organismes biologiques peuvent en avoir plusieurs. L'approche informatique idéalise traditionnellement le chromosome en le réduisant à un vecteur de gènes alors que la molécule d'ADN possède une configuration dans l'espace (ou structure tertiaire) qui évolue dynamiquement et conditionne l'activation des gènes. Nous emploierons indifféremment les termes de chromosome, individu ou vecteur de gènes qui dans le langage particulier des algorithmes génétiques désigne le même concept.
- L'indice de qualité, aussi appelé indice de performance ou valeur, est une mesure affectée à un chromosome indiquant sa capacité en tant que solution à résoudre le problème. Si, à titre d'exemple, nous cherchons le minimum de la parabole $f(x) = x^2$, et que nous disposons d'une solution composée des 3 individus suivants: -1, 0 et 2. Alors 0 se verra attribué l'indice de qualité le plus élevé, suivi de -1 et de 2

respectivement.

- La fonction d'évaluation est la formule théorique qui permet de calculer l'indice de qualité d'un chromosome.
- Population: Un groupe d'individus artificiellement ou naturellement générés.
- Génération: Une génération d'individus désigne une population à un instant donné t du cycle d'évolution des populations.

En pratique, un chromosome est généralement représenté par un vecteur de bits (représentation binaire) ou par un vecteur de réels ou d'entiers (représentation réelle). Nous reviendrons (section 2.7) sur les caractéristiques liées à ces deux types de représentation.

2.5 Fonctionnement d'un GA

Un algorithme génétique élémentaire consiste à faire évoluer une population d'individus possédant des caractéristiques que l'on soumet à une série d'opérateurs génétiques.

Initialement, une population d'individus (*chromosomes*) est aléatoirement générée (mais peut aussi être crée artificiellement par le concepteur). Chaque individu

de la *population* (qui est une solution possible) recevra une note (*indice de qualité*) comme résultat de son évaluation, ce qui engendre un classement parmi les membres de la population. Les opérateurs génétiques sont alors appliqués aux individus afin de créer une nouvelle population statistiquement meilleure que la précédente (en utilisant par exemple la somme ou la moyenne des indices de qualité des individus comme critère de comparaison). Ce processus est réitéré un nombre de générations suffisant pour obtenir un certain niveau de qualité de la population. Bien que simpliste d'un point de vue biologique, ces algorithmes sont suffisamment puissants pour produire de robustes et efficaces mécanismes de recherche adaptative.

2.6 Les opérateurs génétiques

Un GA de base utilise 3 principaux opérateurs génétiques qui sont: la sélection, la mutation et le *crossover*. Un quatrième opérateur appelé "reproduction" et beaucoup moins fréquemment rencontré dans la littérature de GAs consiste tout simplement à dupliquer un individu dans la population, donc à copier ses caractéristiques, c'est-a-dire ses gènes.

2.6.1 La sélection

Le mécanisme de sélection dans un GA n'est rien d'autre que le processus qui favorise la sélection des meilleurs individus de la population sur lesquelles seront appliquées les deux autres opérateurs génétiques pour produire la nouvelle génération.

Pression de sélection. On appelle *pression de sélection*, le degré avec lequel les meilleurs individus sont favorisés: plus la pression de sélection est élevée, plus les meilleurs individus sont favorisés. Il existe divers schémas de sélection pour les GAs. Le schéma idéal serait simple à coder et efficace. La pression de sélection a une importance capitale sur l'évolution de la valeur globale de la population à travers les générations. Le taux de convergence d'un GA est en grande partie déterminé par la pression de sélection. Les algorithmes génétiques sont capables d'identifier les solutions optimales (ou proches) sous un large éventail de la valeur de la pression de sélection [16]. Cependant, si la pression de sélection est trop basse, le taux de convergence sera lent, et le GA prendra inutilement plus de temps pour trouver la solution optimale. Tandis que si la pression de sélection est trop haute, il y a une plus grande chance que le GA converge prématurément vers une solution locale sous-optimale.

2.6.2 La mutation

La mutation est une altération aléatoire d'un ou plusieurs gènes d'un individu. La figure 3 illustre l'effet d'une mutation d'un gène sur un chromosome. Il y a plusieurs types de mutations, mais l'idée fondamentale est que la progéniture reste identique à l'individu parental à l'exception d'un changement au niveau du trait de caractère de l'individu par l'opérateur. Autrement dit, la mutation représente une "promenade aléatoire" dans le voisinage d'une solution particulière. Tout comme dans la nature, une mutation peut aussi procurer à l'individu d'un GA un avantage en terme de sélection, et son caractère aléatoire peut permettre au GA d'échapper à un

maximum local de l'espace de recherche. Le gène modifié peut provoquer un accroissement ou un affaiblissement de la valeur globale de la solution que représente l'individu.

Figure 3: La mutation d'un géne sur un chromosome

2.6.3 Le crossover

Le *crossover* (on retrouve aussi dans la littérature les appellations suivantes: enjambement, croisement ou hybridation) est la transposition informatique du mécanisme qui permet, dans la nature, la production de chromosome qui héritent partiellement des caractéristiques des parents. La figure 4 exhibe la principale propriété du *crossover*: l'information génétique globale est préservée, les gènes étant simplement transférés d'un individu à l'autre. C'est le *crossover* qui permet l'exploitation, par recombinaison, des "briques de bases" sélectionnées au cours des générations passées.

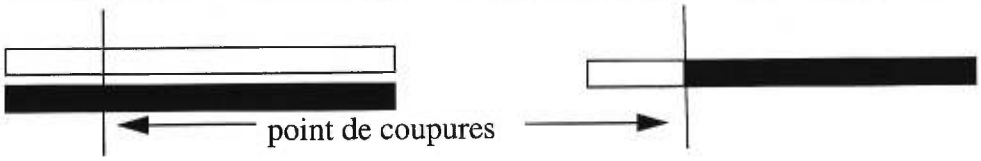

Figure 4: *Crossover* ou enjambement

La version de *crossover* la plus simple est celle à un seul point de coupure. Il s'agit de fixer une même position (point de coupure) sur 2 chromosomes parents sélectionnés dans la population (nous verrons plus loin la méthode de sélection des chromosomes), au niveau de laquelle les chromosomes sont sectionnés. Une partie aléatoirement sélectionnée de l'un des parents est fusionnée avec la partie adjacente de l'autre parent (ainsi que l'illustre la figure 4), pour former le chromosome enfant.

Il existe aussi d'autres variations de *crossover* parmi lesquelles on retrouve:

- Le *crossover* à 1, 2 ou n (n < taille du chromosome) points de coupures formant 1 ou 2 enfants. La figure 5 exhibe deux *crossovers* à 1 et 2 points de coupures formant deux enfants.

Les deux points qui vont suivre présente un type de *crossover* faisant partie des mécanismes dits de recombinaisons multi-parentales:

- Le *crossover* par balayage. Introduit dans [17], ce *crossover* fonctionne en couplant p chromosomes parents pour former un seul chromosome enfant. Le j-ième gène du chromosome enfant est sélectionné (sélection aléatoirement uniforme ou biaisée) parmi les j-ième gènes des p parents.

- Le *crossover* diagonale. Ce type de croisement génétique forme p chromosomes enfants à partir de p chromosomes parents. Le *crossover* diagonale sélectionne $(p-1)$ points de coupures distincts résultant en p segments de chromosome chez

chacun des p parents et compose p chromosomes enfants en prenant les segments des chromosomes parents selon la "diagonale". On peux noter que pour p = 2, cela correspond exactement au *crossover* classique à un point de coupure formant 2 enfants à partir de 2 parents.

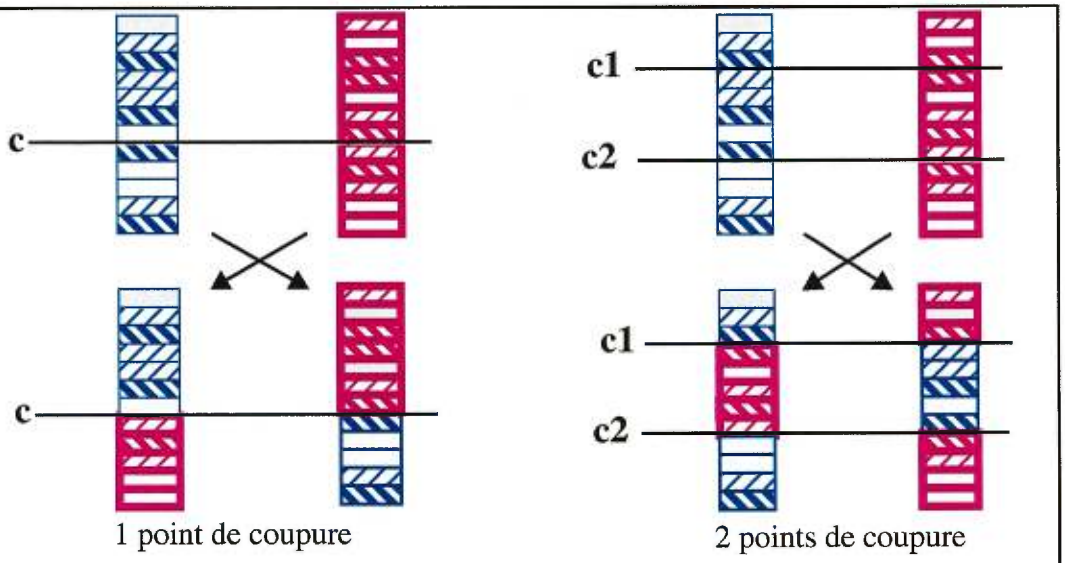

Figure 5: Crossover à 1 et 2 points de coupures.

Sélection des chromosomes pour le croisement. La méthode de sélection naturelle la plus couramment employée pour l'algorithme génétique de base est dite "méthode de la roulette de casino". Chaque chromosome occupe un secteur de la roulette dont l'angle est proportionnel à son indice de qualité. Ainsi, un chromosome considéré comme bon aura un indice de qualité élevé, ainsi qu'un large secteur de la roulette, et par conséquent aura plus de chance d'être sélectionné. L'algorithme présenté à la figure 6 décrit les étapes de cette méthode de sélection.

1.Additionner les indices de qualité de tous les membres de la population ;

2.Tirer un nombre n au hasard entre 0 et la " qualité totale ".

3.Retourner le premier membre de la population dont l'indice de qualité, additionné à celui des membres qui le précèdent dans la population, est supérieur ou égal à n.

Figure 6: Sélection par la méthode de la roulette de casino

2.6.4 L'application des opérateurs génétiques

Plusieurs techniques coexistent pour appliquer les opérateurs génétiques sur une population. La façon la plus répandue (étant celle que nous avons choisie pour notre expérience) est l'application séquentielle des opérateurs dans l'ordre suivant: sélection des parents -> *crossover* -> mutation des enfants. On retrouve d'autres manières d'application des opérateurs génétiques dont la méthode de la sélection aléatoire de l'opérateur qui est illustré dans l'organigramme d'un algorithme génétique exhibé à la figure 8.

2.6.5 Autres opérateurs génétiques

La diversité des opérateurs génétiques n'est limitée que par l'imagination de leur créateur, mais leur efficacité ne peut être discutée que par l'expérimentation. Certains chercheurs iront jusqu'à douter de l'utilité du *crossover* dans le mécanisme des algorithmes génétiques appliqués à certains problèmes. En effet des comparaisons

empiriques sur certains problèmes d'optimisation de fonctions ont montré que la méthode évolutionnaire sans croisement fonctionnait mieux! ([18]).

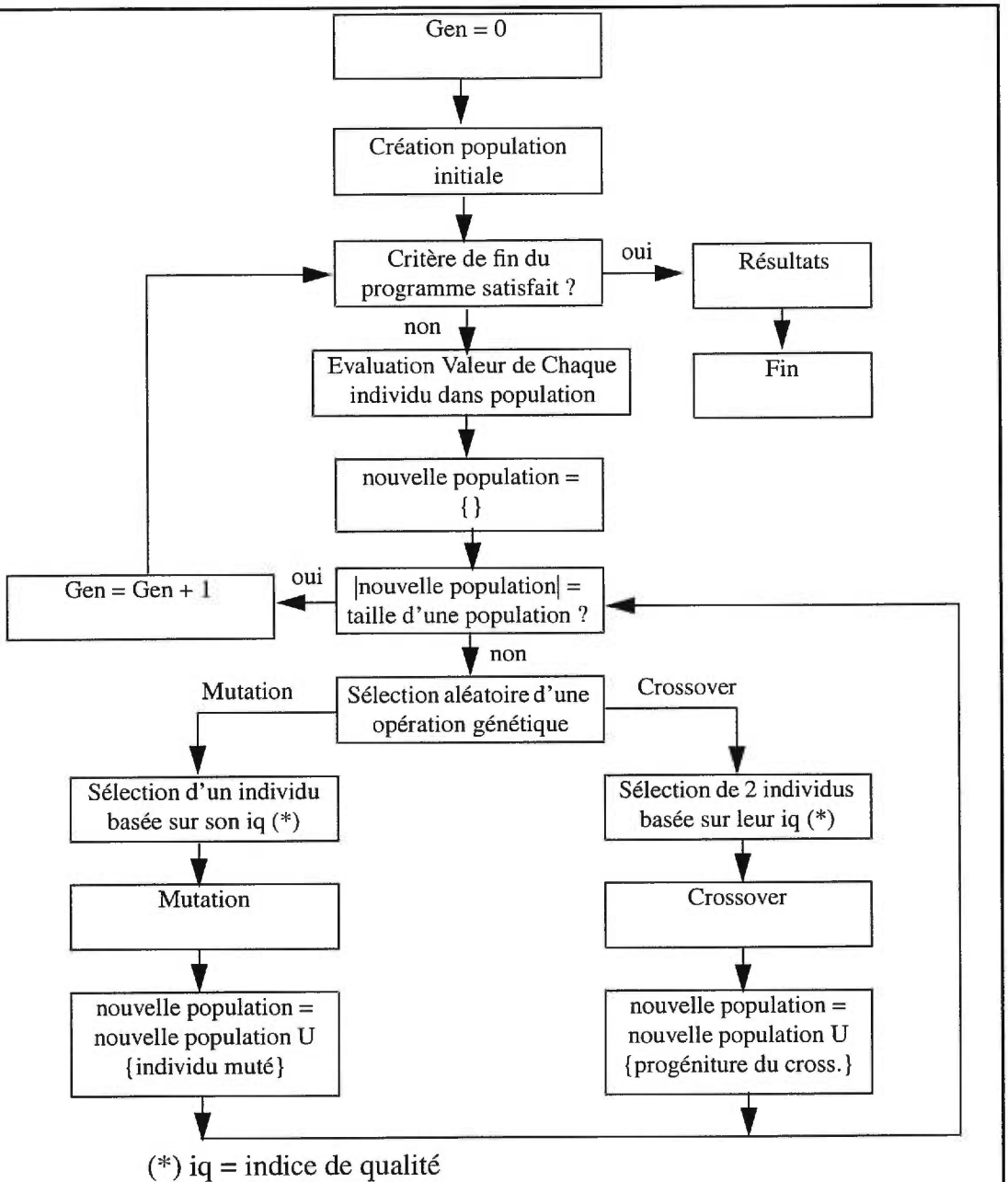

Figure 7: Organigramme d'un algorithme génétique utilisant la sélection aléatoire de l'opération génétique

2.7 Codage d'un individu

Lors de l'application d'un algorithme génétique à un problème particulier, une phase importante est la représentation ou le codage d'une solution. Ce sera en effet sur base de cette représentation que les 3 opérateurs précédents seront appliqués aux populations.

2.7.1 Exemple avec représentation binaire

A titre d'exemple, considérons le problème très simple suivant: Chercher le maximum (ou une solution approchée) d'une fonction du type y = f(x), où x est un entier positif appartenant à l'intervalle [0, 63]. Une façon de modéliser le problème pour le résoudre au moyen d'un GA est d'utiliser la représentation binaire. Dans cette représentation (introduite par John H. Holland [15]), les gènes sont des bits et les chromosomes sont des chaînes de bits. Ce choix de représentation facilite l'application de transformations génétiques telles que le croisement ou la mutation (en effet il aurait été difficile de faire un croisement de deux solutions décimales) et rend l'algorithme génétique indépendant du problème traité. Ceci représente l'avantage majeur de la représentation binaire. Cependant cette représentation demande un effort de calcul supplémentaire: l'espace des solutions entières décimales potentielles doit être transposé dans un espace de solutions binaires en entrée de l'algorithme, et la solution obtenue en sortie doit être reconvertie en une solution réelle afin de pouvoir être interprétée. La figure 7 illustre l'encodage binaire des solutions potentielles.

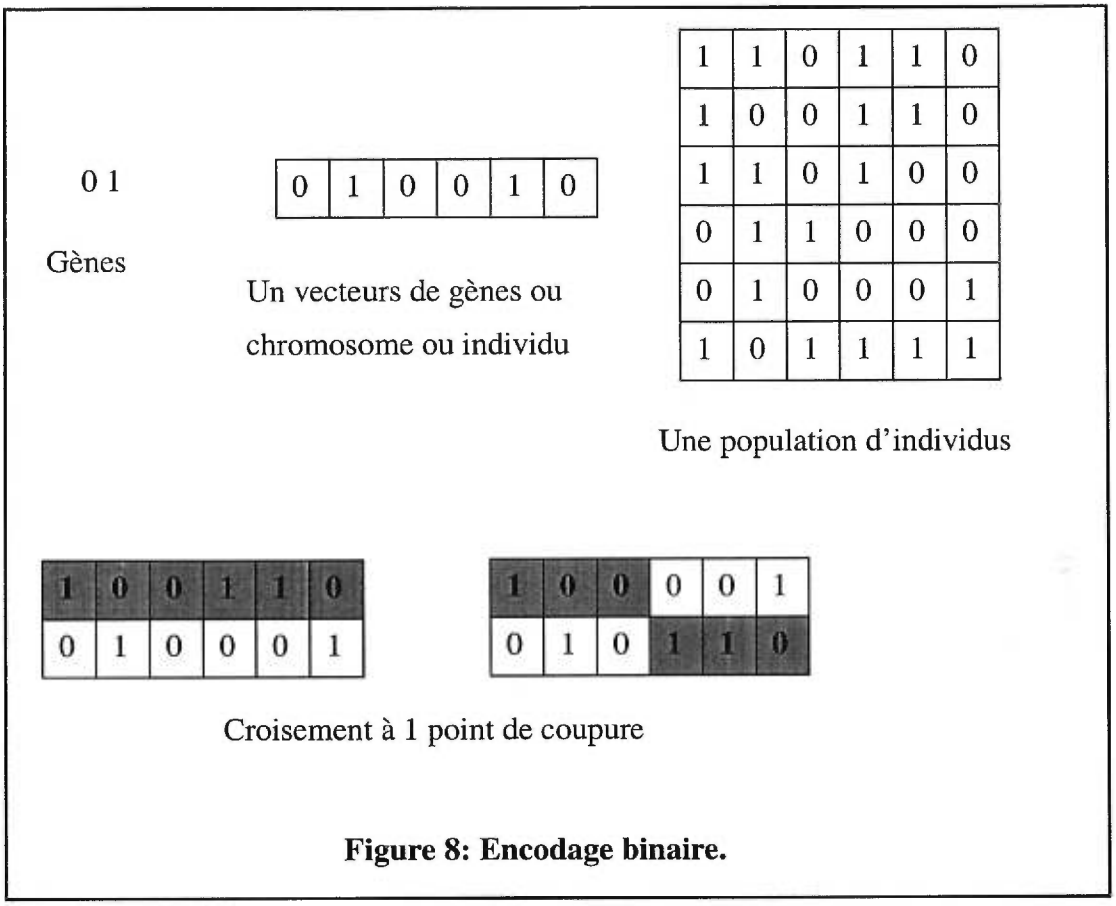

Figure 8: Encodage binaire.

La population de départ se composera de nombres tirés au hasard dans l'intervalle où nous recherchons le maximum qui seront convertis en vecteur de bits pour former les individus. L'indice de qualité d'un individu sera donné par la fonction d'évaluation f appliquée à l'individu. Chaque génération propose une solution au problème qui est l'individu de la population possédant le plus haut indice de qualité. La qualité de la solution de la génération initiale est évidement très médiocre (statistiquement) puisqu'elle a été choisie au hasard. L'algorithme génétique va tirer parti des meilleurs éléments à sa disposition pour fabriquer de toutes pièces un nouvel ensemble d'éléments répondant en moyenne de mieux en mieux au problème. La taille

de la population fait l'objet d'un choix crucial (section 2.8.1) pour le succès de l'expérience. Il est important de noter l'aspect isomorphe essentiel du choix de la représentation qui permet à l'image d'une population par les opérateurs génétiques d'appartenir à l'intervalle de définition des solutions potentielles du problèmes ([0, 63] dans l'exemple).

2.7.2 Inconvénients avec la représentation binaire

Nous avons déjà vu à la section précédente que les chromosomes binaires sont convertis en décimal avant chaque évaluation. Un inconvénient plus grave de la représentation binaire réside dans son incapacité à traiter des problèmes à plusieurs variables (ce qui n'était pas visible par le problème de l'exemple précédent dont chaque solution potentielle est représentée par une variable unique). Avec ce type de représentation, les problèmes multi-variables sont ramenés à un problème mono-variable par concaténation des inconnues en un seul chromosome. A chaque évaluation, la chaîne de bits résultante est alors découpée en autant de sous-chaînes qu'il n'y a d'inconnues; ces sous-chaînes sont ensuite converties en nombres réels pour être appliquées à la fonction d'évaluation. Ces opérations de conversion sont coûteuses en temps-machine, et sont répétées un grand nombre de fois à chaque génération. Nous discernons ici les limites de la représentation binaire.

La représentation réelle élimine toutes les opérations de conversion. Dans ce type de représentation, un chromosome est un n-uplet de nombres réels ou entiers,

chacune des composantes correspondant alors à une inconnue du problème. Nous ne donnerons pas d'exemple illustrant cette représentation car notre méthode de vérification l'utilise.

2.8 Les paramètres d'un GA

La structure d'un algorithme génétique est suffisamment souple pour qu'il puisse s'adapter sans trop de difficultés à différents types de problèmes. Cependant les paramètres qui le modélisent peuvent varier en nombre et en nature d'un problème à un autre. On retrouve néanmoins 3 paramètres communs à tous les algorithmes génétiques:

- la taille de la population détermine le nombre d'individus dont elle dispose. Il convient de bien choisir la valeur de ce paramètre qui a une part de responsabilité déterminante dans la qualité des solutions au problème à résoudre. Une taille de population trop faible peut empêcher le GA d'exploiter le potentiel génétique de certaines solutions qualitativement intéressantes ou peut encore limiter sa recherche si la fonction d'évaluation à optimiser possède de nombreux maxima locaux.

- la probabilité de mutation que subit chaque gène lors de la reproduction.

- la probabilité de *crossover* qui détermine la fréquence à laquelle les hybridations entre individus vont avoir lieu.

Ces deux derniers paramètres sont corrélés. Alors que la mutation permet l'apparition de nouveaux gènes, le *crossover* diffuse les gènes existants lors du renouvellement de la population. Cet équilibre entre l'exploration par mutation et l'exploitation par *crossover* doit être soigneusement respecté car un taux de mutation trop élevé entraîne la destruction de gènes avant qu'ils n'aient eu la chance d'être assemblés par *crossover* afin de former des structures valables. De l'autre côté, si le taux de *crossover* est excessif, la population sera uniformisée trop rapidement et la population convergera alors prématurément vers un type d'individu probablement sous-optimal. On ne parle alors plus de la survie du plus apte, mais de la survie du plus médiocre.

2.8.1 Le réglage des paramètres d'un GA

Par sa difficulté, le réglage des paramètres d'un GA, est souvent assimilé à de la magie noire (la littérature des GA utilise le terme de *Black Art*). Deux types d'approches sont possibles afin de tenter de résoudre ce problème:

- Mettre ces paramètres sous le contrôle d'un super-système adaptatif, un autre GA. Ce type de tentative pose un problème supplémentaire: le paramétrage du super-système.

- Coder dans le chromosome lui même des coefficients qui vont intervenir dans l'activité des opérateurs génétiques. On peut introduire un facteur de lien -- une sorte de colle, comme l'ont implémenté J. D. Schaffer et A. Morishima [19] --

entre les gènes d'un individu afin de produire, par sélection naturelle, de robustes associations de gènes.

- Il est aussi possible ([20]) d'associer aux chromosomes une carte des points de *crossover* possibles et d'utiliser des opérateurs de *crossover* spéciaux. De la même manière, la mutation peut être contrôlée par l'association, à chaque allèle, d'un coefficient de mutation.

- Nous avons utilisé dans notre méthode une technique basée sur une analyse de sensibilité autour des paramètres propres à notre expérience que nous avons considérés comme critiques pour le succès de la recherche par l'algorithme génétique. Nous reviendrons plus en détails sur cette méthode au chapitre qui va suivre.

- La probabilité de croisement ou de mutation peut être évolutive au cours du fonctionnement du GA.

2.9 La convergence prématurée de la population et ses solutions

L'un des problèmes que l'on rencontre fréquemment lors de l'utilisation d'un GA est la convergence prématurée de la population vers un optimum local de l'espace de recherche. La solution à ce problème n'est pas triviale puisque'on s'en rend compte que lorsque l'on connaît la forme de l'espace de recherche, c'est à dire que l'on sait que l'on a raté l'optimum global. Dans d'autres cas, il se peut que l'espace de

recherche possède plusieurs optima globaux alors que le GA ne converge que vers l'un d'entre eux bien qu'il soit souhaitable d'obtenir toutes les solutions possibles au problème que l'on cherche à résoudre.

Il existe différentes recettes qui permettent de limiter l'uniformisation rapide de la population.

2.9.1 Application des mecanismes de croisements multi-parentaux

Le caractère prématuré de la convergence des populations de solutions lors de l'utilisation des GAs est empiriquement atténué ([17, 21]) par l'utilisation de croisements multi-parentaux (section 2.6.3).

2.9.2 Les procédures de sélection

Lors de la phase de reproduction de la population, la procédure de sélection classique consiste à utiliser l'indice de qualité (iq) d'un individu pour lui attribuer une probabilité de sélection (méthode de la roulette de casino, section 2.6.3) définie par:

$$P_{\text{sélection}} = \frac{\text{iq}}{\sum_{\text{I}} \text{iq}}$$

Afin de réduire la vitesse de convergence de la population, une première solution consiste à sélectionner un premier parent en fonction de $P_{\text{sélection}}$ et à choisir l'autre parent de manière totalement aléatoire.

2.9.3 Transformation linéaire: linear scaling

Il est également possible de calculer $P_{\text{sélection}}$ en transformant l'indice de qualité iq d'un individu. Cette transformation vise à diminuer artificiellement l'écart entre les bons individus en utilisant une équation linéaire du type:

$f_{\text{transformée}} = af_{\text{originale}} + \text{b}$

2.9.4 La fragmentation de la population

Il est également possible d'utiliser un ensemble de petites populations au lieu d'une population unique. Dans cette optique, si l'une de ces populations converge prématurément, cela n'affecte en rien la recherche effectuée par les autres populations. De plus, nous pouvons faire migrer d'une population à l'autre les individus localement performants. Bien évidemment la fréquence de ces migrations doit être suffisamment faible pour éviter de faire converger prématurément l'ensemble des populations vers un même type d'individu. Chaque population utilise la même fonction d'évaluation et produit, si l'espace de recherche le permet, une espèce d'individu spécifique.

2.9.5 L'unification des individus: le SSGA

Etant donné que pour certains problèmes la fonction d'évaluation est la partie du GA la plus coûteuse en temps de traitement, un nouveau type de GA est étudié en détail dans [22, 23]: le *Steady State Genetic Algorithm* ou *SSGA*. Dans le SSGA, à chaque cycle, deux individus sont choisis pour la reproduction. Les opérateurs de

crossover et de mutation sont identiques à ceux du GA classiques, mais les descendants nouvellement créés ne sont introduits dans la population que s'ils diffèrent d'individus déjà existants dans la population. Si cette méthode (que nous utilisons dans notre expérience) permet d'éviter l'évaluation d'individus dupliqués, elle préserve également la diversité génétique de la population.

2.10 Les GA et le calcul parallèle

Lorsque la taille de la population est élevée, le temps de convergence du GA, lorsqu'il est traité par une machine séquentielle, augmente au delà du raisonnable. D'autre part, dans un GA classique, le calcul de la valeur d'un individu ne dépend que des caractéristiques contenues par celui-ci. En conséquence, on peut envisager de paralléliser ce calcul.

2.10.1 Un GA pour une machine SIMD

Une machine SIMD (Figure 9) consiste en une matrice de processeurs élémentaires (ou PEs, Processing Elements en anglais) interconnectés qui exécutent simultanément les instructions envoyées par un contrôleur séquentiel.

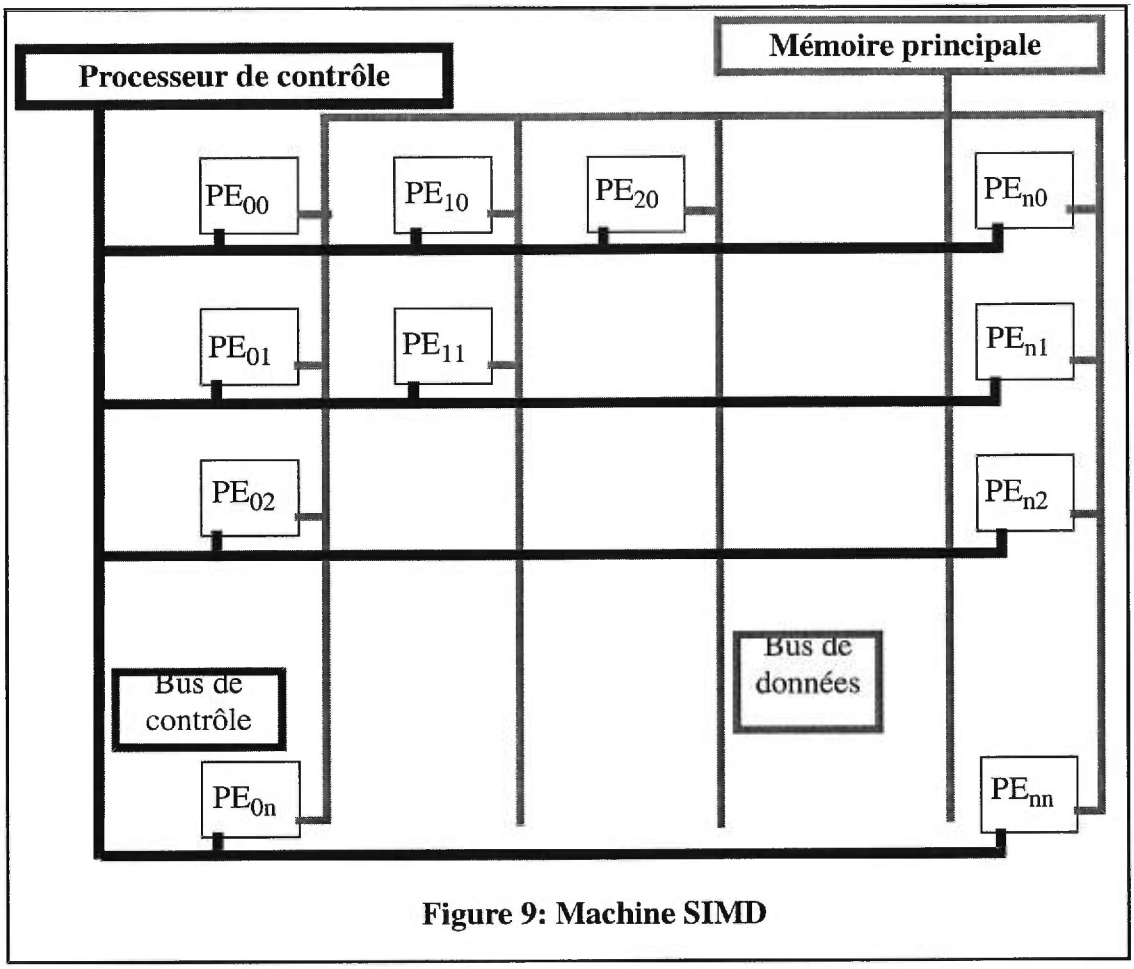

Figure 9: Machine SIMD

Piet Spiessens et Bernard Manderick de l'U.L.B (Université Libre de Bruxelles) ont implémenté un GA sur une machine massivement parallèle SIMD [24, 25], le DAP (Distributed Array Of Processor) un Algorithme Génétique parallèle à fin grain (ou FGA, Fine Grained Genetic ALgorithm). Le DAP est composé d'une grille torique de processeurs élémentaires qui traitent chacun un individu du GA.

L'algorithme implémenté par ces deux auteurs sur le DAP est:

(en parallèle) POUR

chaque PE pe de la grille:

Initialisation:

génère aléatoirement un individu dans pe.

Evaluation:

évalue l'individu de pe.

REPETER

Pré-sélection:

// Cette procédure "purifie" la population en remplaçant les individus inaptes (à faible indice de qualité) par la duplication d'individus plus aptes.

Sélectionner, en fonction de l'indice de qualité, un individu voisin (dans la grille) qui deviendra l'individu contenu dans pe.

Crossover:

Choisir aléatoirement un individu du voisinage de pe. Effectuer le *crossover* entre cet individu et celui de pe en fonction d'un taux de *crossover* et stocker l'un des résultats du *crossover* dans pe.

Mutation:

Changer aléatoirement les caractéristiques de l'individu pe en fonction du taux de mutation.

Evaluation:

Calculer la valeur de l'individu de pe.

JUSQU'A

ce qu'un résultat suffisamment satisfaisant soit produit.

L'intérêt de l'approche est qu'un DAP 510 -- sur lequel cet algorithme a été implémenté -- possède une grille de 32x32 processeurs, ce qui permet de traiter en parallèle une population de 1024 individus. Il est également important de noter que cet algorithme tire parti de l'architecture du DAP en exploitant l'efficacité des connections entre PEs voisins. Pour une longueur de génotype l, la comparaison des complexités d'un GA comportant n individus avec un FGA dont le voisinage comprend s individus est significative:

	GA séquentiel	**FGA**
Compléxité	**O(n.logn + n.l)**	**O(s.logs + l)**

Etant donné la nature locale des interactions entre processeurs, il n'y a pas de convergence prématurée mais plutôt une lente diffusion des caractéristiques des individus au sein de la grille qui permet la formation progressive de grappes d'individus, correspondant à l'exploration de différents points de l'espace de recherche.

2.10.2 Le calcul sur machine MIMD

Il est également possible de tirer parti d'une architecture MIMD (*Multiple Instruction Multiple Data*). Ainsi, un GA parallèle nommé PGA sur un réseau de transputers a été développé dans [26]. Le PGA est dédié à la minimisation de fonctions dont les solutions peuvent s'exprimer sous la forme d'un vecteur de nombre réels. La représentation binaire des solutions traitées par le PGA est tout simplement celle des nombres flottants que la machine utilise pour ses calculs arithmétiques.

2.10.3 Quel cadre pour la parallélisation des algorithmes génétiques

Encore une fois, nous nous trouvons confrontés à une situation dans laquelle il nous est impossible de déterminer à priori une architecture de réseau de processeurs qui sera adaptée au traitement de tous les problèmes. Idéalement cette architecture doit dynamiquement se reconfigurer en fonction de la nature du problème traité. Cette capacité d'auto-organisation, pour un réseau de processeurs, si elle est pour le moment utopique en terme d'implémentation effective, sera d'autant plus nécessaire qu'augmentera le nombre de processeurs mis en jeu. Car s'il est encore possible

d'adapter à la main l'architecture d'un réseau constitué d'une dizaine de processeurs à un problème spécifique, il devient impossible de spécifier une architecture ad hoc si le nombre de processeurs devient très grand. Certes, la Connection Machine CM2 de Thinking Machines [27] possède un système de communication permettant de relier aisément deux processeurs arbitraires, mais ce système n'est pas encore capable de détecter des régularités dans le transfert de données en cours d'exécution du programme afin de planifier dynamiquement le réarrangement des connections entre processeurs et la migration des processus dans le réseau.

2.11 Conclusion

Les algorithmes génétiques peuvent être mis en oeuvre dans un grand nombre de problèmes en conservant la structure de l'algorithme génétique simple. Ils donnent souvent de bons résultats là où une méthode classique s'avère inefficace ou difficile à mettre en place. Ce sont généralement des algorithmes très coûteux en temps de calcul mais nous avons vu qu'il était possible de tirer profit du parallélisme qui leur est intrinsèque (e.g., la recherche simultanée de plusieurs solutions) pour en concevoir des implémentations parallèles efficaces. En revanche on est confronté à un problème sérieux de choix de la paramétrisation optimale de l'algorithme (taille de la population, probabilités de croisement et de mutation, etc...). Enfin, il est habituellement impossible de savoir si la solution obtenue est bien la meilleure, ou si une autre solution peut être trouvée en laissant la population évoluer pendant un

certain nombre de générations. Des embryons de solutions pour ces problèmes existent (ou font l'objet de recherche) mais malheureusement les solutions empiriques présentent encore un rapport qualité/temps d'expérimentation supérieur. Les GA constituent néanmoins une approche pleine d'avenir.

L'introduction des GA comme outil d'aide à la vérification de design de circuits électroniques est une pratique encore assez récente et relativement peu utilisée. Cependant, parmi les techniques de vérification qui les utilisent on peut citer: la vérification de certaines propriétés de protocoles de communication sous certaines conditions de stress du trafic du réseau [28, 29] et la génération de vecteurs de test [30, 31]. Dans les sections qui suivent, nous présentons notre méthodologie de vérification qui utilise la simulation comme outil de base, munie d'un système de pilotage par un algorithme génétique pour l'exploration de l'espace de vecteurs d'entrées au circuit vers des milieux où la couverture du design atteinte par la vérification est optimisée.

Après avoir passé en revue les caractéristiques essentielles des algorithmes génétiques, nous allons voir dans le chapitre qui suit comment nous appliquons ces algorithmes à notre méthode de vérification de designs.

CHAPITRE TROIS

Méthodologie et organisation du système

3.1 Introduction

Notre système de simulation aléatoire guidée est structuré en 4 modules de base (Figure 10), chacun étant constitué d'un ou de plusieurs sous-modules: le module de design du circuit (MD), le module d'environnement du circuit (ME), le module de vérification des propriétés du circuit (MP) et le module de contrôle général de la Simulation (CS). L'activité des signaux d'entrées/sorties échangés entre ME et MD et/ ou internes à MD est analysée par MP pour la vérification des propriétés du circuit. CS utilise l'information dont dispose MP pour gérer l'ensemble du déroulement de la simulation. Avant de présenter la méthodologie de notre expérience, nous allons débuter ce chapitre par une description détaillée du contenu et du fonctionnement de chacun des modules de base.

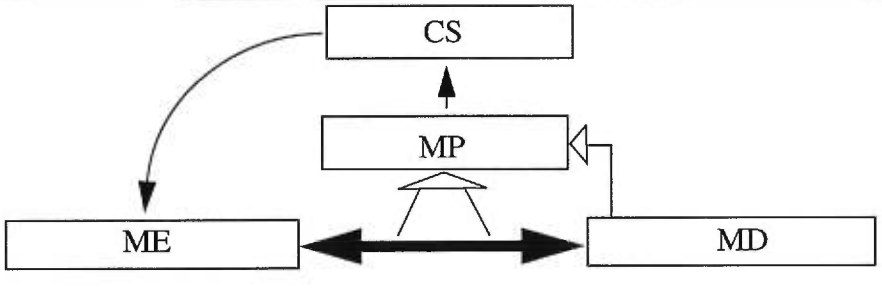

Figure 10: Organisation générale du système en modules.

3.2 Le Design (MD)

Le module de design du circuit (MD) représente le circuit logique synchrone à vérifier, modélisé au niveau transfert de registres (RTL) et décrit en langage HDL (Verilog ou VHDL).

3.3 L'environnement du Design (ME)

Le module d'environnement de vérification du circuit (ME) est un modèle abstrait non déterministe de l'environnement d'utilisation de MD. Le choix non déterministe des transitions/actions possibles dans ME est gouverné par un ensemble de n variables aléatoires. Par conséquent, ME contient n générateurs de nombres pseudo aléatoires (GNPA), chacun étant contrôlé par une fonction de distribution de probabilité différente paramétrisée par un ou plusieurs arguments. ME doit se conformer le plus fidèlement possible au comportement général que l'environnement réel du design pourrait potentiellement avoir. Ce qui pourrait amener à envisager un partitionnement en plusieurs environnements indépendants, chacun étant utilisé dans des simulations différentes. Certaines précautions doivent cependant être prises afin que l'union de toutes les partitions couvre le mieux possible le comportement prévu.

3.3.1 Le vecteur de paramètres

Soit m_j, $1 \leq j \leq n$, le nombre de paramètres caractérisant le GNPA j, le vecteur de paramètres $V = \{p_{11},\ldots, p_{1m_1},\ldots,p_{j1},\ldots, p_{jm_j},\ldots, p_{n1},\ldots, p_{nm_n}\}$ constitue l'élément de base du chromosome de notre algorithme génétique. L'interaction MD <-> ME est schématisée à la figure 11.

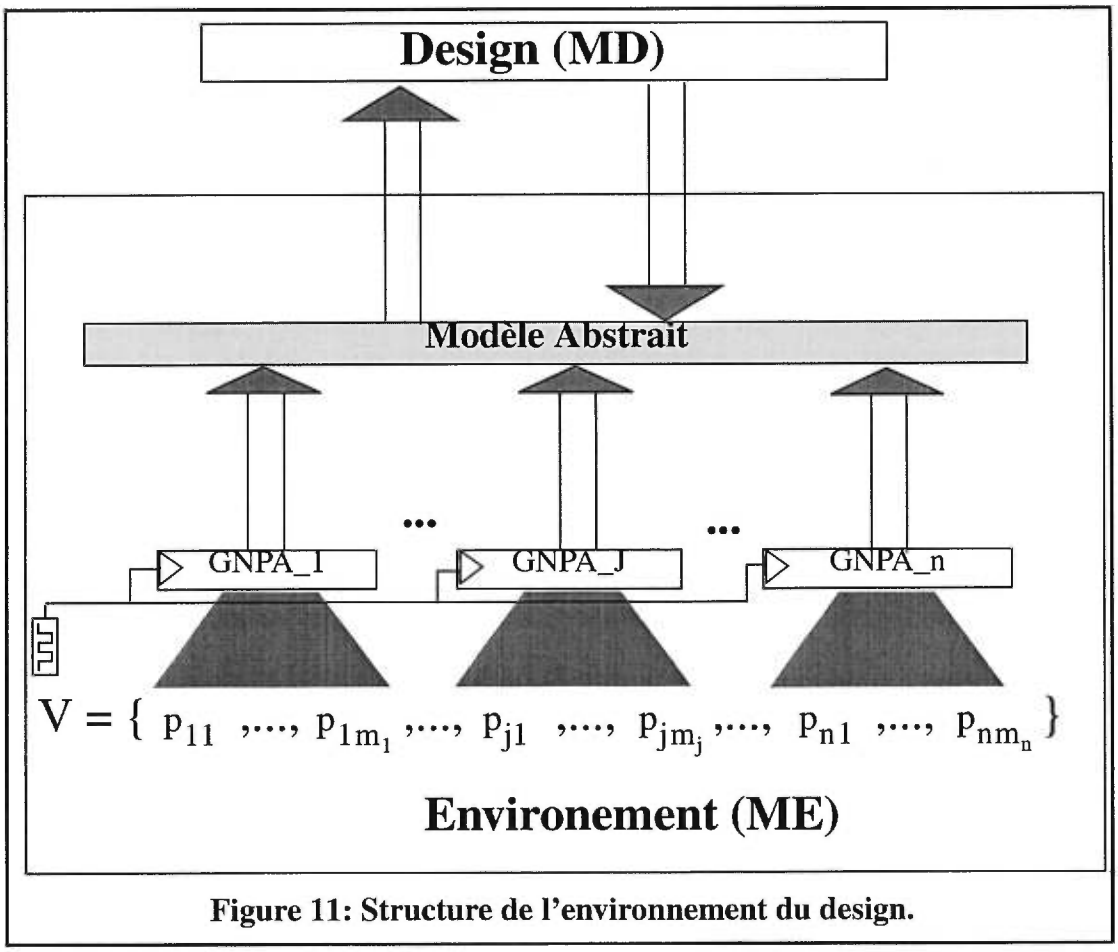

Figure 11: Structure de l'environnement du design.

A titre d'exemple, considérons un circuit dont le modèle non déterministe est composé de 3 GNPAs, les 2 premiers étant contrôlés par des lois de distribution normale ($N(\mu_1,\sigma_1)$,$N(\mu_2,\sigma_2)$, respectivement), et le troisième par une loi de distribution exponentielle ($E(\lambda)$). Chaque vecteur de paramètres pour cet exemple aura la forme suivante: $V = \{\mu_1, \rho_1, \mu_2, \rho_2, \lambda\}$.

3.3.2 Le chromosome ou individu d'une population

Maintenant, si nous associons une durée de simulation d (correspondant à un nombre fixe de cycles d'horloge (L)) à un vecteur de paramètres V, nous obtenons un *chromosome* de l'algorithme génétique (nous utiliserons aussi le terme: *individu* d'une population génétique) qui s'identifie facilement au déroulement d'une simulation de longueur L, dirigée par la distribution de probabilités générée par le vecteur de paramètres V. Une population d'individus ou de chromosomes sera donc composée d'un ensemble de (V, L), chacune simulant pendant une durée fixée par L, un échange séquentiel différent entre MD et son environnement, de signaux contrôlés par une loi de distribution caractérisée par le vecteur de paramètres V.

3.3.3 Simulations partielles

Notre méthode de vérification par simulation est donc fragmentée par un ensemble de (populations de) fractions (temporelles) de simulations caractérisées par un couple (V, L), que nous désignerons: *simulations partielles (V, L)*, par souci de distinction avec le terme général de *simulation* qui fait référence à l'outil utilisé par la

méthode de vérification dans son ensemble. Notons que nous utiliserons dans notre expérience la même valeur de L pour tous les individus d'une population, et cela pour toutes les populations dans une simulation. Ce qui veut dire que deux individus dans une population ne se distingueront que par leur vecteur de paramètres V. Nous verrons plus tard comment il devient possible par l'entremise des signaux à caractère non déterministe (c'est à dire du vecteur de paramètres V) d'avoir un contrôle direct sur la couverture de vérification atteinte par l'ensemble de la simulation.

3.4 Les propriétés (MP)

Pendant le déroulement de chaque simulation partielle de longueur L, le système (MD - ME) est soumis à une vérification portant sur certaines propriétés qui doivent obligatoirement être respectées par le Design pendant la simulation, et plus précisément par les valeurs des signaux transitant par les ports d'entrés/sorties de MD et ME et/ou des signaux internes à MD et ME. MP a une importance capitale dans l'organisation de notre système car la quantité de propriétés du design observées est l'indicateur clé de la qualité de notre méthode de vérification. En effet plus la couverture de propriétés du design atteinte sera large, plus élevée sera la probabilité de découverte d'erreurs, c'est à dire de non respect du design vis-à-vis des contraintes qui conditionnent son bon fonctionnement.

Pour que la vérification soit la plus complète possible, ces propriétés doivent couvrir le maximum de fonctionnalité du Design. Cette tache est loin d'être évidente en particulier lorsque l'on utilise comme outil de vérification la simulation qui, fait bien connu, n'offre aucune garantie sur la dépossession exhaustive de fautes dans un système. La non constatation de "bug" même après une simulation intensive d'un design ne signifie malheureusement pas la non existence d'anomalies dans ce design (*non completeness*). L'emploi de techniques formelles comme "*model-checking*" apportent une solution à ce problème (cf introduction) qui n'est malheureusement pas suffisante lorsque l'on utilise - comme dans notre cas - la couverture de propriétés de design comme objectif du modèle de vérification. En effet, un *bug* peut échapper à l'attention de "*model-checking*" si l'ensemble des propriétés spécifiées ne fait pas référence à la partie défectueuse du comportement du système induite par le *bug*. A ce propos, une étude réalisée dans [32] propose une métrique de couverture pour estimer la complétude d'un ensemble de propriétés vérifiées par "*model-checking*". Une extension intéressante de notre projet serait d'appliquer une métrique similaire à un ensemble de propriétés vérifiées par simulation. Ceci ferme la parenthèse ouverte sur l'importance de la qualité de l'ensemble des propriétés spécifiées dans MD. Nous allons maintenant, dans la section qui suit, présenter le type d'implantation choisi pour la spécification des propriétés dans notre méthode.

3.4.1 Les Vérificateurs

Le module de vérification des propriétés du circuit (MP) renferme un ensemble de processus parallèles qui sont modélisés en machines à états finis, implémentant

chacun un vérificateur ayant pour charge d'assermenter le bon déroulement d'une propriété spécifique du design (MD). Une propriété se présente généralement sous la forme suivante: $H \Rightarrow C$, où H est un détecteur de séquences (prédicat de précondition), et C une formule en logique temporelle (prédicat de conclusion). Chaque processus observe l'activité des signaux aux ports d'E/S (et/ou internes) de MD et ME, en attente de l'apparition de la satisfaction de la précondition de la propriété qu'il vérifie. Si la précondition est détectée, le processus entre dans l'état d'attente de l'apparition de la conclusion, et y reste jusqu'a ce qu'elle soit observée vraie ou fausse.

3.4.2 Les compteurs des vérificateurs

A chaque vérificateur de propriétés, on associe deux compteurs: *le compteur de déclenchements* de la propriété indique le nombre de fois où la précondition H de la propriété a été satisfaite et *le compteur d'erreurs* de la propriété qui est activé lorsque la conclusion C se révèle fausse (F) alors que l'hypothèse H a été préalablement évaluée à vrai (V). Il faut noter que deux situations peuvent se produire en cas de détection d'erreurs au niveau d'une propriété: Si la propriété est jugée fondamentale pour le design, l'erreur est déclarée fatale et la simulation prend fin. Dans le cas où la violation de la propriété par MD est d'un degré de gravité moindre, le compteur d'erreurs est incrémenté et la simulation peut continuer. Certaines propriétés n'imposent pas de contraintes au niveau du nombre de cycle d'horloge entre le déclenchement (H) et la conclusion (C). Dans un tel cas, un maximum de délai de

réponse (time out) est déclaré à l'initialisation au delà duquel la propriété est considérée comme étant erronée et le compteur d'erreurs est incrémenté (ou la simulation est arrêtée).

Les compteurs de déclenchement agissent donc comme indicateurs de la couverture atteinte par la simulation et sont utilisés (nous verrons comment plus loin à la section 3.5.4) par l'algorithme génétique pour guider la simulation vers des espaces de plus en plus riches en activation des propriétés que doivent vérifier le Design.

3.4.3 Un exemple de propriété

Considérons l'exemple simple de la propriété P suivante:

Si à un instant t: ($i_1 = 1$ et $i_2 = 0$) et à t+1: ($i_3 = 0$), alors il faut qu'à t+2: ($o_1 = 1$ ou $o_2 = 1$).

Soit $H_1 = i_1 \wedge \neg i_2$; $H_2 = \neg i_3$; $C = o_1 \vee o_2$.

$$H^{(t)} = (H_1^{(t)} \wedge H_2^{(t+1)}) ; \; P^{(t)} = (H^{(t)} \Rightarrow C^{(t+2)}) .$$

Initialement, le processus responsable de la vérification de P est dans l'état EP_00. Lorsque l'hypothèse H_1 est vraie (= 1), il passe dans l'état EP_01 et surveille la valeur de l'hypothèse H_2 au top d'horloge suivant. Si H2 est observée fausse (= 0), le processus revient dans l'état initial EP_00. Sinon, le processus passe dans l'état EP_10 et examine la valeur de la conclusion C au top d'horloge suivant, alors que le

compteur de déclenchement de la propriété est incrémenté. Le processus revient alors dans l'état EP_00 et dans le cas ou C n'est pas vérifiée, le compteur d'erreurs est incrémenté. Ceci donne lieu à la représentation en machine à états finis de la figure 12.

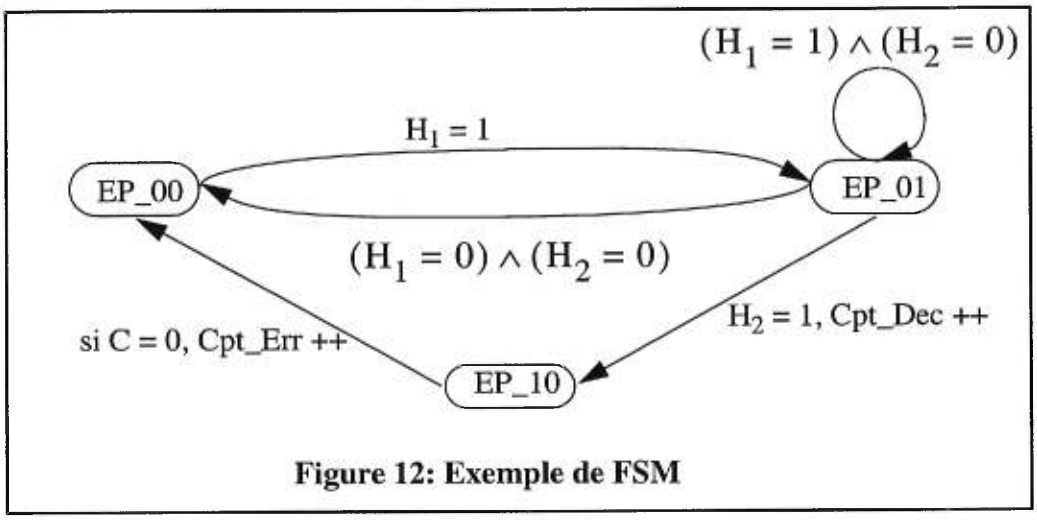

Figure 12: Exemple de FSM

Le pseudo-code correspondant peut alors s'écrire de la façon suivante:

Initialisation:

Cpt_Dec = 0; Cpt_Err = 0; EP = EP_00;

Affectation:

$H_1 = (i_1 \wedge \neg i_2)$; $H_2 = \neg i_3$; $C = (o_1 \vee o_2)$;

A chaque top d'horloge positif durant la simulation partielle:

dans le cas où:

début cas

EP == EP_00,

```
                si H1 == 1 alors EP = EP_01;

        EP == EP_01,
                si H2 == 1 alors
                        EP = EP_10;
                        Cpt_Dec = Cpt_Dec + 1;
                sinon si H1 == 0 alors EP = EP_00;

        EP == EP_10,
                EP = EP_00;
                si C == 0 alors Cpt_Err = Cpt_Err + 1;
    fin cas.
```

Il est important de faire remarquer qu'avec cette représentation, certaines activations de propriétés ne paraîtront pas dans la quantité indiquée par le compteur d'activations de propriétés, car elles n'auront tout simplement pas été prises en compte par le processus de vérification. En effet lorsqu'un processus de vérification d'une propriété est lancé, une autre activation de la même propriété ne pourra être détectée que lorsque le processus reviendra à son état initial (EP = EP_00, autrement dit qu'il ait complété un cycle de vérification). Dans l'exemple précédent, un tel défaut pourrait survenir dans le cas suivant: à t, EP = EP_00 et $H_1 = 1$; à t+1, EP = EP_01, $H_2 = 1$ et $H_1 = 0$; à t+2, EP = EP_10 et $H_1 = 1$; à t+3, EP = EP_00, $H_1 = 0$ et $H_2 = 1$. Une apparition de H n'aura pas été détectée par la MEF. Dans cet exemple simple, il n'y a qu'un seul cas de défaillance, mais ces cas peuvent se multiplier lorsque l'exemple se complique. On pourrait, pour palier à ce défaut, implémenter plusieurs vérificateurs de

la même propriété (c'est à dire initialiser plus d'une variable d'états: EP_i). La question serait alors: combien?

Nous savons maintenant que le vecteur de paramètres (cf 3.3.1) contrôle une simulation partielle entre MD et ME tout en étant évalué dans MP. La prochaine section introduit le module qui gère l'interaction de MD, ME et MP ainsi que la génération et l'évolution des vecteurs de paramètres de l'ensemble de la simulation.

3.5 Le contrôle général de la Simulation (CS)

C'est dans ce module (CS) que sera implémenté l'algorithme génétique sous-jacent à notre méthode de vérification. Il contrôle toutes les phases de développement et d'évolution des populations génétiques, de la première génération (initialisation) à la dernière (fin de la simulation). CS communique avec ME et MP (voir figure 10). Dans ce module, l'information recueillie par MP est exploitée pour transformer une population d'une génération antérieure à une génération postérieure, laquelle sera utilisée par ME pour l'exécution des simulations partielles.

3.5.1 Détermination des paramètres de l'algorithme

Dans un premier temps, il convient d'affecter aux constantes qui paramétrisent l'algorithme génétique les valeurs adéquates. Il s'agit de:

- max_génération. Le nombre maximal de générations de la population de la simulation détermine le temps maximal au delà duquel la simulation est arrêtée. Ce temps peut ne pas être atteint dans le cas où le niveau de couverture observé est jugé suffisant par l'expérimentateur.

- nb_individus. Une population est composée d'un nombre fixe d'individus.

- L. Chaque simulation partielle dure pendant un temps (L) déterminée par un nombre fixe de cycles d'horloge.

- mutation. La mutation (cf chapitre 2, section 4) est un événement aléatoire qui se produit sur un individu avec une probabilité donnée par ce paramètre. En d'autres termes, *mutation* donne le nombre moyen d'individus mutés dans chaque population.

- ps. La pression de sélection est le degré avec lequel les meilleurs individus sont favorisés (section 2.6.1). Ce facteur permet de ne pas appliquer les opérateurs génétiques sur toute la population mais seulement sur une partie sélectionnée. L'algorithme de sélection est très simple: Les ps individus ayant le plus petit indice de qualité ne sont pas candidats aux opérations génétiques pour la formation de la nouvelle génération.

D'autres constantes qui paramétrisent l'algorithme génétique n'ont pas encore été introduites et seront présentées un peu plus loin dans la section.

La recherche de la solution optimale de l'ensemble de ces paramètres génétiques est un sujet qui fait actuellement l'objet de beaucoup d'intérêt de la part des chercheurs dans ce domaine (cf chapitre 2, section 5). Dans notre expérience, nous avons opté pour une analyse de sensibilité des paramètres clés (dont nous donnerons plus de détails dans le chapitre 4), afin de déterminer expérimentalement la valeur adéquate des paramètres qui optimise la solution.

Nous pouvons maintenant exécuter le processus génétique qui débute par la génération aléatoire des individus de la première génération (3.5.2) avant d'entrer dans le cycle de l'évolution des populations (3.5.3).

3.5.2 L'initialisation

Cette phase d'initialisation consiste à former une population de départ. Ceci peut se faire de façons différentes selon les besoins et la structure du système. On peut générer aléatoirement dans un domaine de valeurs prédéfini tous les vecteurs de paramètres de la population. Il est également possible de sélectionner de façon précise les coordonnées des tous les vecteurs de la population initiale. Ceci est utilisée lorsque par exemple on essaie de découvrir la présence éventuelle d'un type d'erreur bien précis dans le Design. Une solution mixte des deux précédentes est également envisageable. Il est important de noter l'importance de cette phase d'initialisation car la population originale contient le bagage génétique de base des générations postérieures et par conséquent l'espace exploré par la simulation ainsi que les

solutions apportées par l'algorithme en sont fortement dépendants. Une fois cette étape d'initialisation achevée, le cycle d'évolution des populations peut commencer.

3.5.3 Le cycle d'évolution des populations

Une population constituée de *nb_individus* simulations partielles -prête à être exécuter - est alors disponible dans le CS. Cette population, aprés avoir été évaluée, sera soumise aux opérateurs génétiques qui produiront la prochaine génération d'individus. Ce processus sera répété jusqu'à obtention d'un niveau satisfaisant de la couverture des propriétés du design.

- Détermination de la valeur des individus

Après l'exécution séquentielle (jusqu'au dernier individu de la population) de chacune des simulations partielles, une valeur ou indice de qualité (dont les détails d'évaluation seront exposés plus tard dans cette section) est affectée à l'individu correspondant.

- Sélection des meilleurs individus

Une fois que tous les *nb_individus* individus ont été évalués, la population est ordonnée par *valeur* croissante et les *ps* (pression de sélection) pires individus (en terme de *valeur*) sont alors évincés de la population. La population restante contient donc les meilleurs individus sur lesquelles seront appliquées les opérations génétiques afin de constituer la prochaine génération d'individus.

- Application des opérateurs génétiques sur la population

Le cross over:

Deux individus sont aléatoirement choisis dans la population de candidats sélectionnés, pour être combinés. Le choix d'un individu est biaisé par sa valeur. On applique la méthode de la roulette de casino (cf section 2.6.3). Si Vi est la valeur de l'individu i, ce dernier sera sélectionné avec la probabilité Pi = $Vi/\left(\sum_{i} Vi\right)$. Ce qui veut dire qu'un individu de valeur plus élevée aura une probabilité plus forte d'être sélectionné. La méthode de *crossover* choisie est le *crossover* biparental à 1 point de coupure, formant deux enfants (cf section 2.6.3). Ce type d'hybridation de deux individus (parents) mène à la naissance de deux individus (enfants), chacun d'eux ayant comme patrimoine génétique une combinaison (différente) du génotype de chaque parents. En d'autres termes, un sous ensemble des coordonnées du premier vecteur de paramètres (parent) sélectionné sera fusionné avec un sous ensemble du deuxième pour former le premier vecteur de paramètres (enfant). Son "frère" sera constitué par la fusion des compléments des deux sous ensembles (parents). Nous avons choisi de façon arbitraire de positionner le point de coupure au centre du vecteur de paramètres (figure 13.a). Ce qui veut dire que chaque vecteur enfant sera constitué de la moitié supérieure d'un vecteur parent et de la moitié inférieure de l'autre. Notons qu'il aurait été interessant de faire une analyse plus approfondie sur la position optimale du point de coupure, mais notre sujet n'étant pas axé sur l'étude des AGs,

nous avons choisi de nous concentrer sur l'analyse des paramètres qui ont une influence plus importante sur l'efficacité de l'algorithme (cf 4.3). Il est important - dans un souci de cohérence - de rassembler les paramètres d'une même distribution, et il convient donc de prendre cet aspect en considération dans le choix de la position des points de coupures.

La mutation:

Lorsque la nouvelle progéniture est crée, chacun des enfants est sélectionné, avec une certaine probabilité (*mutation*), pour subir l'étape de mutation génétique. Si l'individu est sélectionné, alors un paramètre du vecteur est arbitrairement choisi (parmi tous les paramètres du vecteur) pour être aléatoirement modifié.

La séquence d'opérations: cross over + mutation est appliquée sur la population d'individus ayant survécu à l'opération de sélection, le nombre de fois nécessaire à former une nouvelle génération de *nb_individus* individus-enfants qui constitueront la nouvelle population courante. Alors que le *crossover* essaie d'optimiser la valeur moyenne des individus à travers les populations, la mutation essaie plutôt d'éloigner les séquences d'un optimum local (états de transitions similaires) et les force à explorer aléatoirement les autres parties de l'espace des séquences d'entrées, et donc d'autres comportements du circuit.

- Répétition du processus avec la nouvelle population

Ce processus est répété de génération en génération (cf figure 13.b) jusqu'à ce que le nombre de générations maximal soit atteint ou alors que la couverture soit satisfaisante à l'expérimentateur.

3.5.4 Calcul de la valeur d'un individu

L'objectif de la méthode est de pouvoir augmenter pendant la simulation la couverture des propriété du design atteinte qui doit être par conséquent directement liée à la valeur d'un individu. L'information sur la couverture du design pourrait être obtenue de diverses sources [6, 5], mais pour cette expérience nous nous limitons à la définir comme l'ensemble des propriétés vérifiées pendant la simulation, couvrant différents aspects fonctionnels du design, et donc différentes régions du code. La couverture est donc exprimée en terme de *compteurs de déclenchements* de chaque propriété. Chaque compteur de déclenchement est doté d'un seuil (S) et l'objectif est atteint lorsque tous les seuils ont été atteint (ou dépassés). Pendant chaque simulation partielle, toutes les propriétés sont vérifiées mais une seule est sélectionnée (propriété active) pour être optimisée. La valeur d'un individu est donnée par l'équation:

valeur d'un individu = valeur du compteur de déclenchement de la propriété sélectionnée pendant une simulation partielle.

A la fin des N simulations partielles de chaque population, si le compteur de déclenchement de la propriété active a atteint son seuil, la propriété dont la valeur du compteur de déclenchement est la plus faible devient la nouvelle propriété sélectionnée pour optimisation. La simulation s'arrête lorsque tous les compteurs sont au moins égaux à leur seuil ou alors que le maximum de temps de simulation permis a été atteint.

3.5.5 pseudo code

Le module de contrôle général de la simulation implémente le pseudo code suivant:

```
Début
  nouvelle_population = Générer_aléatoirement_population_initiale;
  tant que (niveau de qualité désiré ou maximum du temps de simulation ne sont pas atteint)
     vieille_population = nouvelle_population;
     nouvelle_population = {};

     boucle-pour (ind = 1 jusqu'a max_population);
        simulation_partielle_avec (individu[ind]);
        evaluer_valeur_de (individu[ind]);
     fin boucle pour

     boucle-répéter jusqu'a |nouvelle_population| = nb_individu;
        (parent_1, parent_2) = selection_aléatoire_biaisée_par_valeur (vieille_population);
```

```
            (enfant_1, enfant_2) = cross_over(parent_1, parent_2);
            si (mute) selection_aléatoire_d'un_parametre_et _mutation (enfant_1);
            si (mute) selection_aléatoire_d'un_parametre_et _mutation (enfant_2);
            nouvelle_population = nouvelle_population U {enfant_1, enfant_2};
        fin boucle répéter
    fin tant que
fin
```

individu est un vecteur ou tableau de taille (nb_gènes), où *nb_gènes* représente le nombre d'entrées dans un vecteur de paramètres (ou individu). *nouvelle_population* et *vieille_population* sont des matrices de dimensions (*nb_gènes*, *nb_individus*), ou *nb_individus* est la taille de la population (ou nombre d'individus dans chaque population). *mute* est une variable booléenne qui exprime le résultat d'un tirage (oui ou non) avec probabilité donnée par *mutation*.

Figure 13.a: Opérations Génétiques.

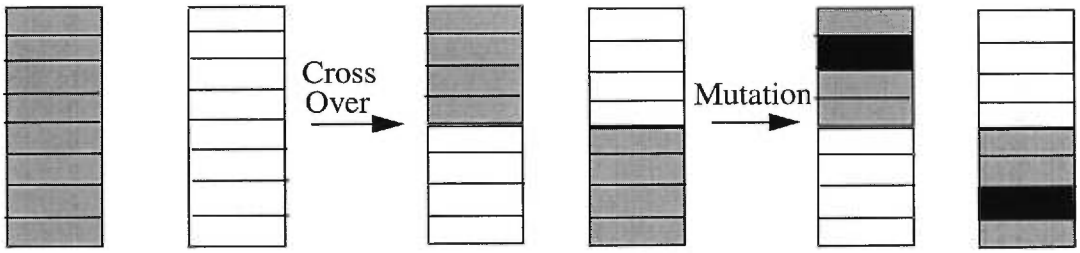

Figure 13.b: Processus Génétique.

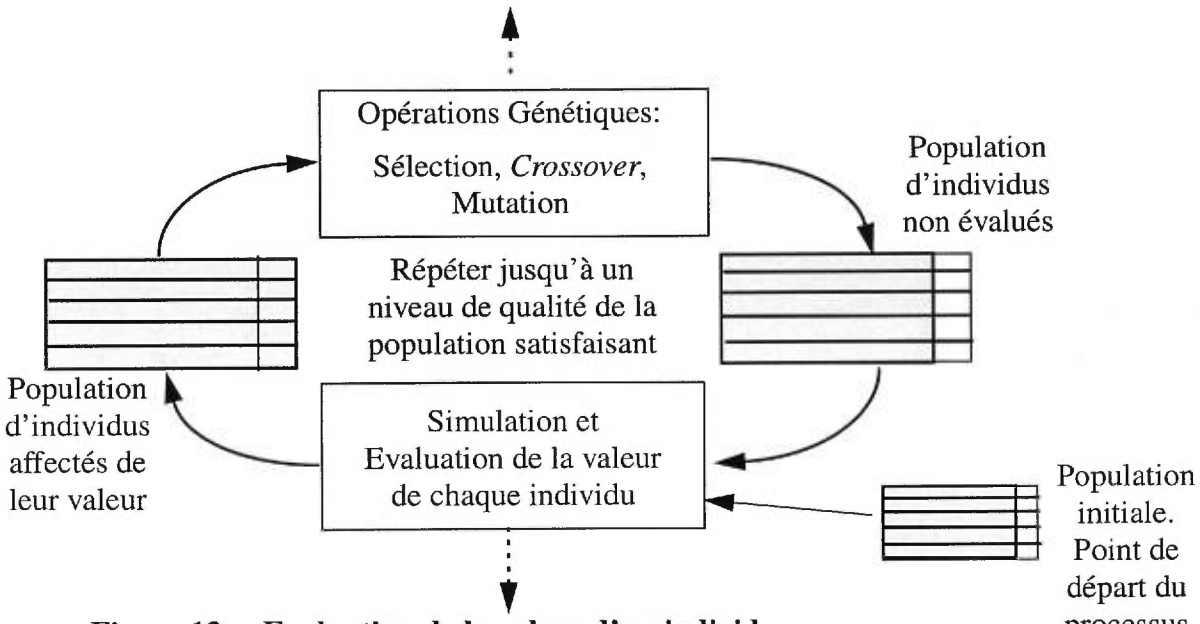

Figure 13.c: Evaluation de la valeur d'un individu.

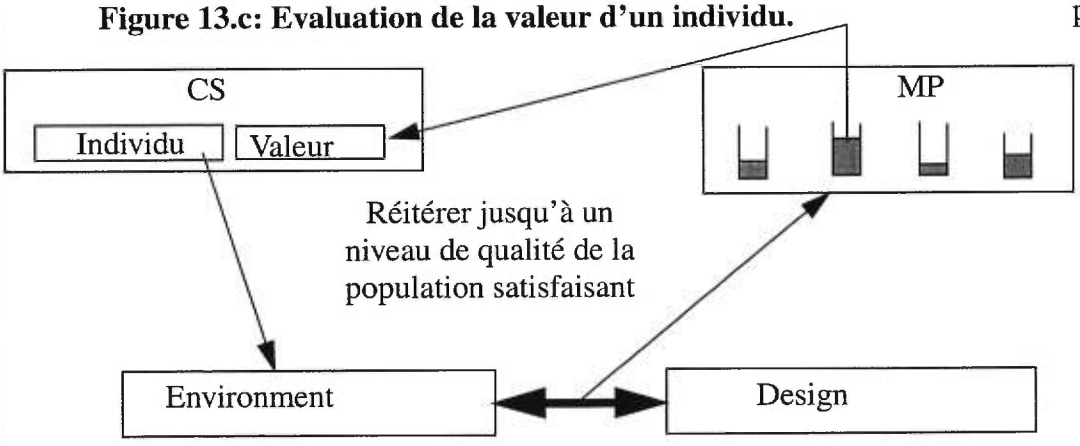

Figure 13: Simulation aléatoire guidée par algorithme génétique

CHAPITRE QUATRE

Résultats Expérimentaux

Dans les quatre premières sections de ce chapitre, nous présentons les résultats obtenus à partir d'une expérience réalisée sur un modèle RTL Verilog inspiré d'une application industrielle d'un Sous-système de Reordonancement de Transactions (SRT) consistant de 2 files de 8 éléments chacune, l'une d'entre elle étant une queue de priorité, d'un port de requête à basse priorité et d'un block de multiplexage de sortie contrôlé par une FSM (machine à états finis) complexe. Nous avons construit un modèle non déterministe de l'environnement du SRT de manière à ce qu'il soit capable de sélectionner toute séquence d'entrées valides, à tout instant, en donnant simplement une valeur particulière à ces ports d'entrées primaires. Au total, la taille du modèle combiné atteint environ 6000 portes équivalentes. Le même modèle a également été analysé par FormalCheck et a présenté 2.00e+07 états atteints pour la propriété 1 décrite plus loin dans le chapitre.

Par souci de protection industrielle, nous ne pourrons pas donner de détails sur le circuit de la seconde application. Cependant nous présenterons en dernière section, quelques résultats expérimentaux observés à partir de ce deuxième modèle d'une taille d'environ 800,000 portes logiques équivalentes.

Les expériences ont été conduites de la façon suivante:

1. On exécute une simulation guidée jusqu'à ce que tous les compteurs sont au dessus du seuil correspondant. On répète plusieurs fois cette expérience en changeant à chaque fois la valeur initiale (*seed*) associée au générateur de nombres pseudo-aléatoires; ce qui a pour effet d'exécuter des simulations différentes par leur population initiale et par conséquent la zone initiale pour l'exploration de l'espace d'état varie pour chaque simulation.

2. Exécuter un certain nombre de simulations purement aléatoires (c'est à dire sans aucun guidage génétique) en utilisant des distributions uniformes. Chaque simulation purement aléatoire consiste à associer à chaque entrée libre du module d'environnement, un générateur de nombres aléatoires de distribution uniforme. Par exemple, en langage HDL Verilog, l'affectation suivante:

ND = $dist_uniform(seed,0,7); // ND étant déclaré comme registre de 3 bits,

a pour effet de retourner à ND une valeur binaire choisie aléatoirement entre 000 et 111. "seed" est la valeur initiale du générateur qui diffère d'une simulation à une autre.

3. Durant chacune des deux types de simulations (génétique et purement aléatoire), on enregistre les valeurs du temps de CPU utilisé et des différents compteurs de propriétés (couverture). Une moyenne calculée sur 10 expériences de chaque type de simulation nous permet de comparer notre méthode à la méthode de simulation traditionnelle.

4.1 Les modèles du Design et de son environnement

Un diagramme en block du SRT et de son modèle d'environnement non déterministe est illustré par la figure 14. Le SRT accepte trois sortes de requêtes de transactions selon la priorité: hautes (H), moyennes (M) et basses (B). H et M sont requises sur les lignes partagées HMReq (Requête pour enfiler), HMTrans_in (le numéro de transaction 0..7), HnM (Haute priorité = 1, Moyenne priorité= 0), Desc (un descripteur sur 2 bits indique la ressource de destination).

La file Médium est une file prioritaire qui sélectionne les objets à enlever de la file, en se basant sur le descripteur de destination Desc et la disponibilité de la ressource de destination indiquée par HMAvail. La file à priorité Haute est une file régulière et les transactions peuvent dépasser celles de la file Médium. Le détachement d'une transaction dans cette file ne dépend pas de la disponibilité de la ressource. Les requêtes de transactions L sont communiquées sur des lignes séparées car elles sont traitées de façon indépendante lorsqu'il n'y a ni transactions H ou M à traiter; Elles servent de processus d'arrière plan.

La principale contrainte sur les transactions H et L est qu'à partir du moment où un numéro de transaction à été soumis par l'environnement à l'entrée HMTrans_in, le même numéro de transaction ne pourra plus être soumis par l'environnement tant qu'il n'aura pas été libéré de la file, sortie par le port HMTrans_out (validé par le

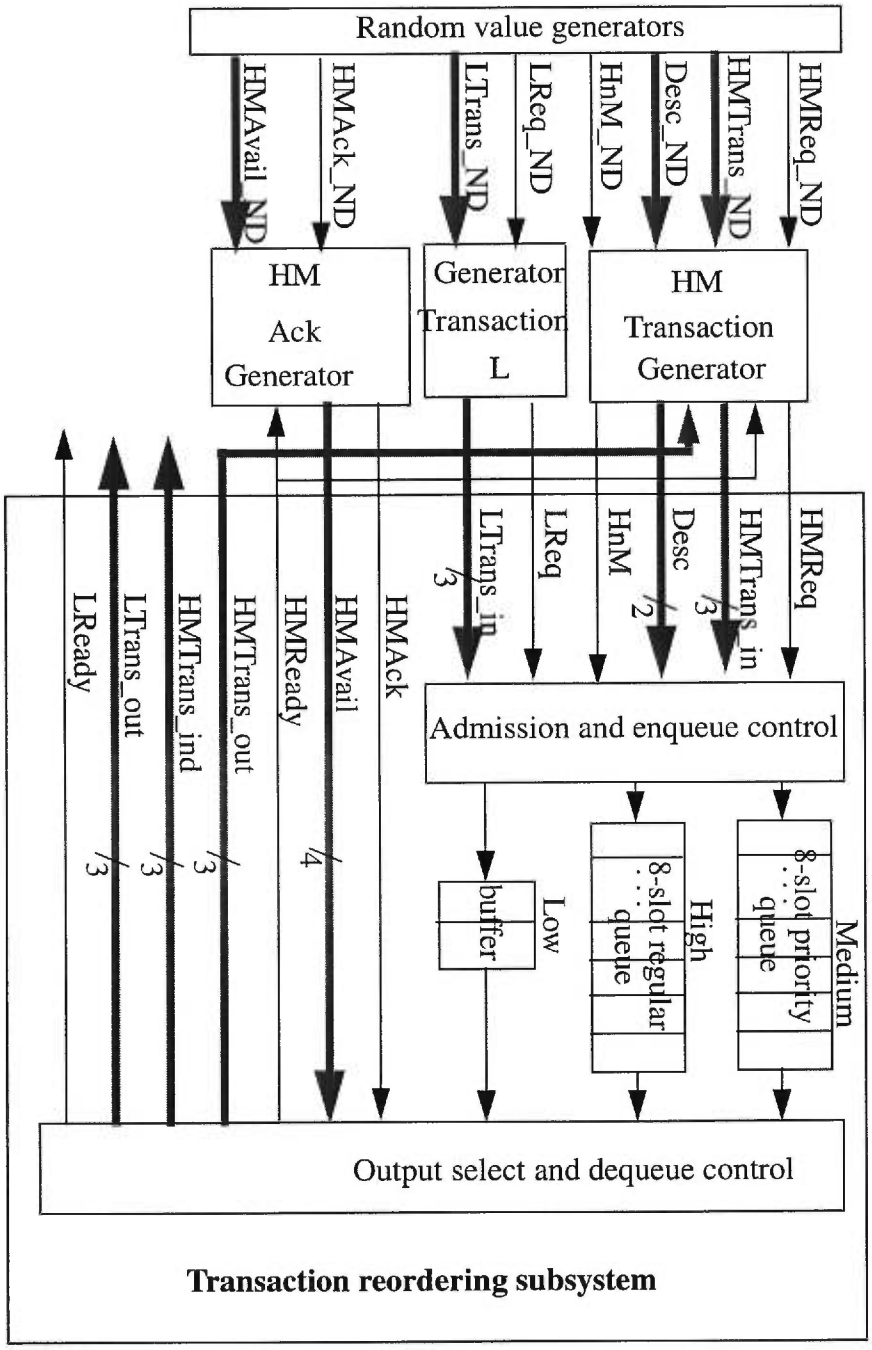

Figure 14: Sous-système de Réordonancement de Transactions et son environnement non déterministe.

signal HMReady), et acquitté par l'environnement au moyen d'une pulsation au niveau du signal HMAck. Par conséquent, le modèle d'environnement doit avoir un procédé de mémorisation des numéros de transactions soumis. Ceci est réalisé au moyen d'une table d'occupation de 8 bits dans le générateur de transaction HM. Selon la valeur de HnM, la transaction est insérée dans la queue correspondante. Il y a autant de positions dans la queue qu'il y a de numéros de transaction, ce qui garantit à tout instant un espace suffisant dans chaque file. Le signal Ack doit être généré au moins 2 cycles après que HMReady ait été observé.

L'environnement est contrôlé par les signaux d'entrée dénotés par xxx_ND, chacun d'entre eux dirigeant aléatoirement l'opération qui lui est associée durant le cycle d'horloge courant. Par exemple, HMReq_ND décide de la soumission d'une requête pour le numéro de transaction généré par HMTrans_ND. Si cette transaction n'est pas déjà en traitement, alors le générateur peut produire (sous contrôle de HMReq_ND) une requête réelle à l'entrée du SRT, tandis qu'aucune requête n'est générée si la transaction est en cours de traitement. Les valeurs de Desc_ND sont directement transférées à l'entrée Desc de SRT.

La décision par LReq_ND d'émettre une requête à basse priorité et le numéro de transaction correspondante donné par LTrans_ND sont générés aléatoirement. Les valeurs de LReq et de LTtrans_in doivent être conservées jusqu'à ce qu'elles soient acquittées par une transition montante provenant du signal LReady.

Chaque signal xxx_ND est associé à un type de distribution aléatoire munie de ses paramètres. Comme ces signaux font référence à des fonctions différentes du SRT, les types de distributions qui leur sont associées sont également différentes. Par conséquent, chaque paramètre doit être interprété selon le type de distribution qui lui correspond. Par exemple, le paramètre associé avec les signaux à 1 bit (HMReq_ND, HnM_ND, LReq_ND, HMAck_ND) donne la probabilité que le générateur correspondant produise un 1 en sortie. Tandis que pour les générateurs de numéro de transaction associés au signal HMTrans_ND à 3 bits, on affecte une probabilité d'occurrence p_i à chaque numéro de transaction i = 0,1,...,7. Ceci résulte en un ensemble de paramètres, chacun représentant la probabilité qu'un numéro de transaction soit produit en sortie par HMTrans_ND, avec $\sum p_i = 1$. De cette façon, nous avons un contrôle direct sur le type et le taux de transactions échangées entre le Design et son environnement.

4.2 Mesure de couverture et paramètres génétiques

Les propriétés établies verbalement dans cette section ont préalablement été exprimées en CTL (*Computation Tree Logic:* [33], [34]) en fonction des signaux d'entrée/sortie primaires du SRT et de l'état d'occupation de la table dans le générateur de transactions HM. Dans une seconde étape, elles ont été converties en machines à états finis, exprimées par des processus en langage de modélisation Verilog. Chaque processus s'assure que la propriété correspondante est vérifiée et

incrémente le compte de déclenchement de la propriété et le compte d'erreurs le cas échéant (cf chapitre 3 section 4). Les processus sont intégrés dans le module de vérification (MP) qui est rattaché au reste du modèle (MD + ME) et au module de contrôle de la simulation (CS) qui dirige le guidage par le support de l'algorithme génétique.

4.2.1 Propriétés

Nous avons sélectionné 4 groupes de propriétés représentatives (Tableau 1). Le module Verilog MP contient 4 vérificateurs (machines à états finis) qui observent les signaux échangés entre MD et ME et qui ont accès à la table d'occupation dans le générateur de transactions HM. Les 2 premiers groupes de propriétés (une propriété pour chaque i) vérifient que l'opération de réservation de la table est correcte. Le groupe de propriétés 3 vérifie le comportement du système relativement aux 2 priorités (H et M) pour tout i et j. Le dernier groupe (4) assure que le numéro de transaction est émis en sortie selon le protocole exact. Il existe beaucoup d'autres propriétés, mais leur(s) précondition(s) sont similaires, et par conséquent leur activation ne fournit pas de renseignements supplémentaires pour l'analyse du comportement de l'algorithme de guidage et furent donc écartées des objectifs de notre étude.

Groupes	Description des groupes de propriétés.
p1	**Pour tout $0 \leq i \leq 7$, lorsqu'un numéro de transaction *i* est soumis avec HMReq = 1 (la précondition), alors l'entrée *i* de la table d'occupation vaut 1 et prend la valeur 0 au prochain cycle d'horloge (la conclusion), mémorisant alors que ce numéro a été soumis au SRT.**
p2	**Dès qu'un numéro de transaction *i* entre dans le SRT, il ne peut pas être soumis de nouveau avec HMReq = 1 tant qu'il n'en est pas ressorti par le port HMTrans_out et n'a pas été acquité.**
p3	**Pour tout $0 \leq i \neq j \leq 7$, lorsqu'une transaction à moyenne priorité *i* est précédée dans le SRT par une autre transaction à haute priorité *j*, alors *j* doit quitter le SRT avant *i*.**
p4	**Lorsqu'une transaction sort par le port HMTrans_out avec HMReady = 1 avant d'être acquittée, alors le même numéro de transaction doit apparaitre sur HMTrans_ind 3 cycles d'horloges plus tard .**

Tableau 1: Groupe de propriétés à vérifier.

Note: Il est important de remarquer que chaque groupe p1, p2 et p4 fait intervenir 8 propriétés (pour tout i, $0 \leq i \leq 7$), alors que p3 en fait intervenir 56 ({i,j}, $0 \leq i \neq j \leq 7$). C'est la raison pour laquelle nous parlons de groupes de propriétés.

4.3 Analyse de sensibilité autour de N et L

Lorsque nous avons à travailler avec des algorithmes génétiques, il est loin d'être évident de choisir les valeurs numériques qu'il faut affecter au paramètres génétiques (cf chapitre 2 section 5) comme la longueur d'une simulation partielle (L), la taille de la population (N), le nombre maximal de générations (max_gen), etc., afin d'optimiser la recherche de solution par l'algorithme. Le succès de toute méthode basée sur un GA dépend fortement d'un choix "astucieux" de ces paramètres. Cette

section présente les résultats de simulations qui nous ont permis d'approcher la (ou les) valeur(s) de L et N qui optimisent les comptes de déclenchement des propriétés. Le nombre de générations maximal utilisé lors de ces simulations préalables est très limité par rapport à celui que nous utiliserons plus tard pour l'expérience de vérification proprement dite. Il est néanmoins suffisant pour nous permettre de comparer la qualité des simulations en fonction du couple (N, L) initial. La méthode de calcul de la valeur des individus (cf chapitre 3 section 5) sera également beaucoup simplifiée. Ces deux mesures nous permettent d'avoir une idée sur la valeur adéquate des paramètres génétiques pour optimiser l'expérience proprement dite, et ceci dans un temps relativement court. Pour ce faire, nous commençons par fixer une valeur de L et exécuter plusieurs simulations avec différentes valeurs de N. Nous répétons alors cette méthode avec différentes valeurs de L. Les résultats sont enregistrés dans le tableau 2 selon les valeurs de (N,L).

Pour chaque entrée du tableau 2, nous exécutons une simulation purement aléatoire et une simulation aléatoire guidée génétiquement. La référence (nature de la valeur des individus) utilisée pour les simulations génétiques est égale au niveau du compteur de déclenchement atteint seulement pour la propriété 3 (qui est la plus difficile à déclencher). Les autres propriétés n'interviennent pas dans la détermination de la valeur des individus.

Pour décrire notre méthode de comparaison de façon plus détaillée, considérons l'entrée du tableau (150, 600) comme exemple. Cette expérience utilise

une population de N = 150 individus (vecteurs de paramètres), chacune correspondant à une simulation partielle de L = 600 cycles d'horloge (ce qui veut dire qu'une population est "génétiquement régénérée après 150 x 600 = 90000 cycles d'horloge.).

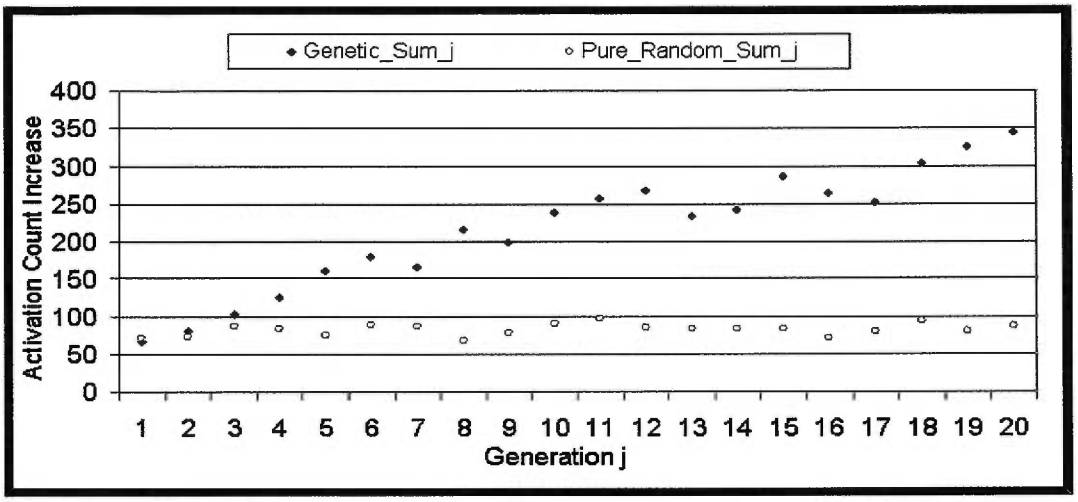

Figure 15: Amélioration de la couverture par un GA pour N = 150 et L = 600

La figure 15 présente 2 graphiques (points par points) qui mettent en évidence l'accroissement de la couverture atteinte pendant la simulation génétique par rapport à celle obtenue durant la simulation purement aléatoire. Le graphique intitulé "Genetic_Sum_j" (série de points en losange plein) représente la simulation guidée génétiquement alors que le graphique intitulé "Pure_Random_Sum_j" (série de points circulaires) représente la simulation aléatoire pure (chapitre 4, p. 58). Pour tracer le graphique "Genetic_Sum_j", nous exécutons une simulation guidée génétiquement (avec la paire de paramètres (150, 600)) et pour chaque génération (en abcisses, allant de la génération 1 (initiale) à la génération max_gen (finale)), nous additionnons les

une population de N = 150 individus (vecteurs de paramètres), chacune correspondant à une simulation partielle de L = 600 cycles d'horloge (ce qui veut dire qu'une population est "génétiquement régénérée après 150 x 600 = 90000 cycles d'horloge.).

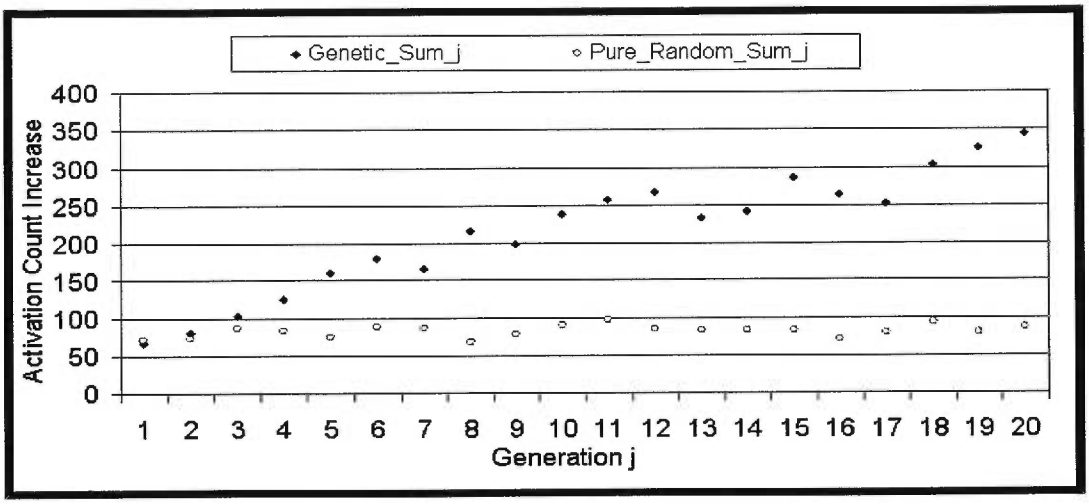

Figure 15: Amélioration de la couverture par un GA pour N = 150 et L = 600

La figure 15 présente 2 graphiques (points par points) qui mettent en évidence l'accroissement de la couverture atteinte pendant la simulation génétique par rapport à celle obtenue durant la simulation purement aléatoire. Le graphique intitulé "Genetic_Sum_j" (série de points en losange plein) représente la simulation guidée génétiquement alors que le graphique intitulé "Pure_Random_Sum_j" (série de points circulaires) représente la simulation aléatoire pure (chapitre 4, p. 58). Pour tracer le graphique "Genetic_Sum_j", nous exécutons une simulation guidée génétiquement (avec la paire de paramètres (150, 600)) et pour chaque génération (en abcisses, allant de la génération 1 (initiale) à la génération max_gen (finale)), nous additionnons les

150 indices de qualité des individus de la population (en ordonnées). Cet ensemble de points peut se représenter mathématiquement de la façon suivante:

$$\text{Genetic\_Sum}_j = \sum_{i=1}^{N} \text{indice de qualité individu (i)} \quad , \quad j \in \{1 .. \text{max\_gen}\} \quad \text{où}$$

max_gen = 20. Pour tracer le graphique "purement aléatoire", nous exécutons une simulation purement aléatoire. Les paramètres génétique L et N n'intervenant pas dans la simulation purement aléatoire, nous procéderons de la façon suivante: après chaque 90000 cycles d'horloge (ce qui correspond au temps de simulation d'une population génétique: N x L), nous enregistrons le contenu du compteur de déclenchement de la propriété 3 (ordonnée des points du graphe) et nous remettons ce compte à zéro. On note que cette valeur correspondrait exactement à la somme des indices de qualité des individus de la population pour une expérience génétique. En d'autres termes, nous comparons sur la figure 15 les contenus (| différence des contenus entre 2 générations successives |) des compteurs de déclenchement (de la propriété 3) des 2 types de simulations après chaque 90000 cycles d'horloges.

Soit $\{ \text{Pure\_Random\_Sum}_j, \; j \in \{1 .. \text{max\_gen}\} \}$, l'ensemble de points qui représente le graphique "purement aléatoire", le rapport:

$$\text{Efficiency(N, L)} = \left(\sum_{j=1}^{\text{max\_gen}} \text{Genetic\_Sum}_j \right) / \left(\sum_{j=1}^{\text{max\_gen}} \text{Pure\_Random\_Sum}_j \right),$$

donne l'entrée (N, L) du tableau 2 (qui vaut 2.54 dans cet exemple).

L \ N	20	50	80	100	150	200	230	250	300
300	1.28	1.45	1.58	2.00	1.78	1.81	2.03	2.01	2.18
600	1.57	1.61	2.24	2.17	2.54	2.72	2.78	3.04	3.04
700	1.76	1.86	2.04	1.74	2.31	2.61	2.78	3.07	2.81
1000	1.64	1.90	2.01	2.65	3.02	3.09	2.75	3.17	2.53

Tableau 2: Analyse de sensibilité de N et L

Les résultats présentés dans le tableau 2 montrent que, avec les restrictions imposées à ces expériences préliminaires d'analyse de sensibilité, le modèle génétique est plus efficace que le modèle aléatoire pure, apportant une augmentation allant jusqu'à un rapport de proportionnalité de 3.17 (Efficiency(250,1000)) du nombre d'activations de la propriété 3 sur seulement 20 générations.

Une analyse de la valeur moyenne de Efficiency(N, L) sur les différentes valeurs de L montre que pour une valeur fixe de N, Efficiency (N, L) aura tendance à accroître avec L. Moyenne(Efficiency(N, L = 300)) = 1.8, alors que Moyenne(Efficiency(N, L > 300)) > 2.3. Nous en concluons (de façon empirique) que les expériences utilisant 300 cycles d'horloges (ou moins) par simulation partielle, bien que favorisant la méthode génétique (Efficiency(N, L = 300)) > 1), ont un taux d'efficacité faible.

Nous pouvons facilement identifier une zone du côté inférieur droit du tableau 2 où sont localisées les valeurs des entrées (N, L) les plus élevées (> 2.60). Parmi ces paires (N, L) à "haut niveau" d'efficacité, nous avons le choix entre réduire le nombre (N) de vecteur de paramètres (individus) dans une population et utiliser des durées de simulation (L) élevées ou alors augmenter le nombre d'individus dans une population et diminuer la longueur de simulation. Le critère décisif ici pourrait bien être le temps total de simulation. Prenons un exemple dans le tableau. Nous remarquons que les entrées (200, 600) et (230, 1000) ont à peu près le même niveau d'efficacité (2.72 vs. 2.75, respectivement). La paire (230, 1000) conduit à une simulation de longueur égale à $230 \times 1000 \times 20(\text{max\_gen}) = 4{,}600{,}000$ cycles d'horloge, alors que la paire (200, 600) mène à une simulation de longueur seulement égale à: $200 \times 600 \times 20(\text{max\_gen}) = 2{,}400{,}000$ cycles. Ce qui veut dire que pour approximativement le même niveau d'efficacité, une simulation guidée pourrait durer deux fois plus longtemps qu'une autre!!! Pour cette raison, le choix N = 200 et L = 600 apparaît adéquat pour l'expérience proprement dite sur le circuit sous vérification.

Les expériences présentées dans cette section n'optimisent que la propriété 3 et comme mentionné précédemment, ne génèrent pas plus de 20 populations. Le but principalement visé lors de ces expériences préliminaires de courte durée, était de nous permettre de faire un choix approprié de la valeur des constantes L et N qui paramétrisent l'algorithme génétique. Ces valeurs obtenues, nous pouvons maintenant exécuter l'expérience principale utilisant notre méthodologie présentée au chapitre 3 et dont les résultats seront présentés à la section suivante.

4.4 Résultats de la simulation

Nous avons exécuté 10 simulations guidées par l'algorithme génétique et 10 simulations purement aléatoires, chacune d'entre elles utilisant une valeure initiale différente pour ses générateurs de nombres aléatoires. Les simulations ont été effectuées sur une station Sun ULTRA SPARC 1 possédant une capacité de mémoire de 52 MB. La simulation guidée requiert environ 7148 secondes (moins de 2 heures) de temps CPU (une moyenne sur les 10 expériences), tandis que la simulation aléatoire pure n'atteindra jamais tous les seuils affectés au compteur de déclenchement des propriétés avant d'être terminée par le nombre maximal de générations préalablement fixé requérant plus de 13200 secondes de temps CPU (plus de 4 heures).

4.4.1 Paramètres Génétiques

Les paramètres utilisés par l'algorithme génétique sont résumés dans le tableau 3.

PARAMÈTRE	NOM	VALEUR
Longueur de simulation d'un individu [cycles d'horloge]	**L**	**600**
Taille de la population	**N**	**200**
Max du nb. de générations	**max_gen**	**45**
Probabilité de mutation	**mutate**	**25%**
Seuil des comptes de déclenchement des groupes de propriétés	**Thresh_P**	**8000**

Tableau 3: paramètres génétiques pour le circuit à 6K portes.

Les tableaux 4 et 5 fournissent plus de détails sur les expériences. Elles sont organisées de la façon suivante:

- I est l'indice numérique de la simulation,
- Tx est le temps CPU requis pour les simulations guidées (x = g) et purement aléatoires (x = r) [secondes],
- Vg est le nombre d'individus (chromosomes) requis pour les simulations guidées. Le produit V x L donne le nombre total de vecteurs d'entrée dans le circuit (un vecteur d'entrée est appliqué au circuit à chaque top d'horloge),
- Vr est le nombre de vecteurs d'entrée (ou top d'horloge) requis pour les simulations purement aléatoires divisé par L. (La division par L permet la comparaison de Vr avec Vg),
- Nyg est le nombre de déclenchements du groupe de propriétés y = 1,..,4 lors de l'approche guidée,
- Nyr est la moyenne du nombre de déclenchements du groupe de propriétés y = 1,..,4 lors de l'approche aléatoire pure,
- P indique les groupes de propriétés qui furent optimisés et dans l'ordre dans lequel ils ont été sélectionnés par l'algorithme (les groupes de propriétés ne figurant pas dans la liste ont atteint leur seuil dans le processus d'optimisation des autres groupes de propriétés),

- μ est la valeur moyenne des valeurs correspondantes dans les tableaux sur 10 simulations,

- Note: le nombre maximal d'individus pour les simulations aléatoires pures a été fixé à 9000 (ce qui correspond à 1,800,000 vecteurs d'entrée).

I	Tg	Vg	N1g	N2g	N3g	N4g	P
1	7990	4825	537	1607	4825	462	p2, p3
2	7844	4737	572	1615	4737	524	p2, p3
3	6993	4102	449	1762	4102	446	p4, p3
4	6990	4221	418	1698	4221	498	p1, p3
5	7829	4728	529	1656	4728	451	p2, p3
6	6172	3727	561	1953	3727	542	p3, p3
7	6526	3941	469	1845	3941	528	p3,p3
8	7687	4642	487	1601	4642	492	p2, p3
9	6804	4109	426	1852	4109	501	p1, p3
10	6644	4012	514	1876	4012	429	p4, p3
μ	7148	4304	496	1746	4304	487	

Tableau 4: Simulations guidées par un algorithme génétique:

Stats	Tr	Vr	N1r	N2r	N3r	N4r
μ	> 14904	> 9000	478	2342	> 9000	513

Tableau 5: Simulations aléatoires pures (moyenne sur 10 experiences):

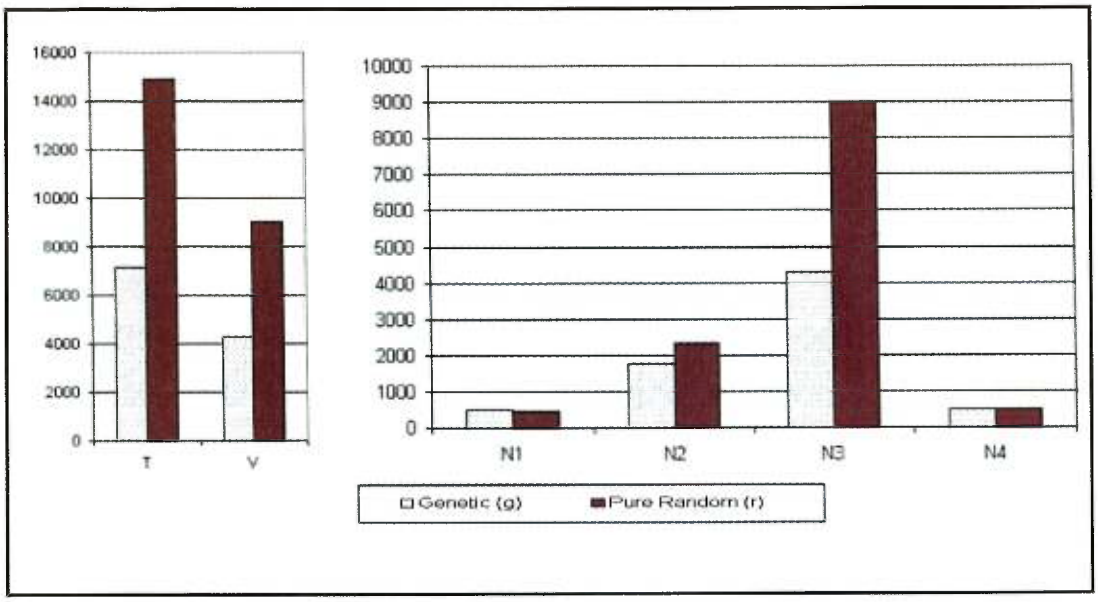

Figure 16: Comparaison de Tx, Vx, Nyx (moyenne sur 10 experiences) entre les simulations génétiques et aléatoires pures

4.4.2 Interprétation des résultats

- La figure 16 montre que pour atteindre le même niveau de compte de déclenchement des propriétés, la simulation aléatoire pure a besoin d'au moins L = 9000 vecteurs d'entrées, ce qui représente presque deux fois plus que la moyenne des valeurs obtenues par les simulations guidées. Ceci veut dire que, par exemple, pour atteindre le même niveau de couverture sur de plus grands designs, la simulation guidée pourrait se dérouler seulement sur 3 jours au lieu de 6 pour la version aléatoire pure.

- Après un choix initial aléatoire, l'algorithme génétique sélectionne toujours la propriété la plus difficile à activer (p3), et alors ajuste les vecteurs de paramètres

des distributions de façon à maximiser son compte de déclenchement plus rapidement que le ferait la simulation aléatoire pure.

- Si le même nombre de vecteurs était appliqué à la simulation purement aléatoire comme à la simulation guidée pour atteindre le seuil d'activation du groupe de la propriété la plus difficile à activer (p3), alors le compte d'activation n'aurait même pas atteint la moitié du seuil des 8000 considéré comme étant le but pour la simulation guidée

- Lorsque nous fixons le paramètre du générateur de nombres aléatoires associé au signal d'entrée HMReq_ND à une valeur élevée (resp. faible), laissant les autres paramètres se faire aléatoirement guider par l'algorithme, nous pouvons observer qu'après un certain nombre de générations, le paramètre HMAck_ND prend aussi des valeurs élevées (resp. faibles). Ceci veut dire que lorsque nous augmentons le taux de transactions soumises au circuit, l'algorithme génétique force le système à diminuer la période de temps qu'il prend pour acquitter les transactions préalablement soumises.

4.5 Simulation et résultats de la deuxième application

Nous avons aussi appliqué notre méthodologie sur un modèle de circuit industriel de télécommunication, d'une taille d'environ 800,000 portes équivalentes, basé sur la technologie SONET (Synchronous Optical NETwork), permettant un transfert de trames (DS1 ou DS3) entre quatre interfaces différentes. Due à la symétrie de notre expérience par rapport aux différents chemins possibles pouvant être empruntés durant le transfert des trames de types différents, nous avons réduit notre

étude au transfert de trames de types DS1 entre deux interfaces dans les deux directions. Durant chaque simulation partielle (cf section 3.3.3), les deux interfaces sélectionnées s'envoient cinq trames successivement. Pendant ce transfert, un microprocesseur externe ordonne des opérations d'accès en lecture sur un bloc de mémoire destiné au stockage de trames DS1 (entre autres). La longueur temporelle entre la fin d'un tel cycle d'opération de lecture et le début d'un autre est aléatoire.

Chaque trame émise est associée à une action particulière choisie aléatoirement parmi un nombre fixe d'actions. Cette sélection aléatoire est pondérée par un ensemble de paramètres qui constitue une partie du vecteur de paramètres (cf section 3.3.1). L'autre partie (deux entrées) du vecteur est constituée du paramètre associé à la décision par le microprocesseur d'un accès en lecture (à chaque top d'horloge succédant à la fin de l'exécution d'un cycle de lecture) et du paramètre associé au choix (également aléatoire) de l'adresse du registre lu.

Les paramètres utilisés par l'algorithme génétique (section 3.5.1) sont résumés dans le tableau suivant:

PARAMÈTRE	NOM	VALEUR
Longueur de simulation d'un individu [cycles d'horloge]	**L**	**5 x (nb. de cycles d'horloge par trames émises)**
Taille de la population	**N**	**30**
Nombre max. de Générations	**max_gen**	**15**
Probabilité de mutation	**mutate**	**.25**

Tableau 6: Paramètres Génétiques de l'application 2

Nous avons procédé à l'éxécution de 5 simulations guidées génétiquement, chacune utilisant une valeur initiale différente pour les générateurs de nombres pseudo-aléatoires, et une simulation purement aléatoire (cf chapitre 4, p. 58), en

utilisant une station Sun SPARC 20 Model 60 avec une capacité de mémoire de 256 MB.

Pendant les simulations, quatre propriétés (cf section 3.4) sont vérifiées mais une seule est ciblée. Cette dernière s'assure par un signal d'acquittement provenant du bloc de mémoire concerné que l'adresse effectivement accédée est celle impliquée dans la requête du microprocesseur.

La durée L de chaque simulation partielle est le produit du nombre de trames envoyées (5) pendant chaque simulation partielle (individu) par le nombre de cycles d'horloges nécessaires à l'envoi d'une trame. Calculé par une méthode similaire à celle employée lors de l'expérience effectuée avec la première application (section 4.3), le facteur d'amélioration (Efficiency (30, L)) de la méthode génétique par rapport à la méthode purement aléatoire est supérieur à trois tandis qu'il était d'environ deux pour le premier circuit.

Nous avons conclu de cette seconde expérience que notre méthode de vérification guidée peut aussi bien s'appliquer sur des circuits de petites tailles que sur des circuits de tailles plus élevées. Cependant cette méthode n'étant pas indépendante de la connaissance sur le circuit (détermination des propriétés, choix des entrées libres associées aux générateurs de nombres pseudo-aléatoires, etc...), le temps de recherche utilisé pour l'adaptation adéquate de la méthode sur le design à vérifier augmente avec la compléxité/taille du circuit. D'autre part, la durée de simulation étant aussi proportionnelle à la taille du circuit, le nombre limité d'expériences que nous avons pu réaliser diminue le degré de fiabilité des résultats obtenus.

CHAPITRE CINQ

Conclusions

Ce mémoire décrit une méthode de simulation aléatoire, guidée par un algorithme génétique pour atteindre son objectif principal qui est l'amélioration de la couverture de vérification du design. Nous la comparons avec celle traditionnellement obtenue par les méthodes de simulation purement aléatoire. Nous présentons une expérience où nous appliquons notre méthode sur un design RTL Verilog d'un circuit constitué d'environ 6000 portes logiques équivalentes. Notre mesure de la couverture est basée sur les niveaux de comptes d'activation de quatre groupes de propriétés que doivent vérifier le design et que nous avons implémentés en machine à états finis (processus Verilog passif). L'objectif était d'élever chacun de ces comptes d'activation jusqu'à une quantité supérieure au seuil préalablement fixé (8000). Les résultats constatés sont encourageants en ce sens qu'ils ont démontrés qu'il est effectivement possible d'influencer la génération aléatoire des séquences de signaux aux entrées d'un design de circuit pour augmenter les comptes d'activation des groupes de propriétés de façon beaucoup plus rapidement qu'avec une simulation purement aléatoire. En la présence d'une propriété difficile à faire activer (p3), le temps de simulation a été diminué de moitié pour atteindre le même niveau de couverture, ou inversement, le compte d'activation de cette propriété atteint le double de celui obtenue d'une simulation purement aléatoire avec le même nombre de vecteur de signaux d'entrées soumis au circuit.

Nous avons également testé notre méthodologie sur un plus grand circuit industriel de 800K portes logiques équivalentes et observé des résultats similaires.

Nos contributions principale ont été:

- L'application des algorithmes dits évolutionnaires sur un problème bien particulier de vérification de design de circuits électroniques.
- L'utilisation originale des paramètres de distributions de nombres aléatoires utilisées par la simulation comme information de base (chromosome) de l'algorithme génétique.
- L'estimation de la couverture de vérification atteinte par le nombre d'activations de différentes propriétés à vérifier par le système, liées à différentes fonctionnalités du système.

Ce travail marque seulement le début d'une plus ample investigation dans ce domaine, incluant des tests sur de plus grands designs. Parmi les questions émanant directement de ce sujet de recherche et qui restent encore sans réponse, nous pouvons mentionner:

- Jusqu'à quel point et comment les connaissances sur la structure du design peuvent aider à choisir les distributions appropriées associées aux entrées libres

du design, la corrélation entre ces entrées, les paramètres génétiques, etc.?

- Observerions nous de meilleurs résultats en modélisant les séquences d'entrées désirées en processus aléatoires et en implémentant les générateurs de nombres aléatoires pour avoir de telles caractéristiques?

- Quelles sont les métriques de couverture appropriée et comment les regrouper? Serait il possible d'utiliser la couverture de code ou la couverture de paire-transition pour guider les simulations?

- Dans l'éventualité où un paramètre de couverture (exemple, un compte d'activation) n'est jamais activé, comment guider les générateurs d'entrées pour cibler ce paramètre? Pourrait-on manuellement inclure des métriques de proximité entre les groupes de propriétés dans l'évaluation de la valeur des individus? Ces métriques pourraient-elles être extraites automatiquement à partir du design et de la structure des propriétés, en sa basant sur les relations de dépendances?

- Comment est-il possible de détecter que l'accroissement du compte d'activation de la propriété courante se stabilise et qu'il est alors possible de passer à une autre propriété avant d'atteindre le nombre maximal de générations ?

- Approfondir l'analyse sur l'algorithme génétique qui supporte la simulation en étudiant la possibilité de parrallélisation du processus et la détermination de la valeur optimale des paramètres tels que le positionnement du (ou des) point(s) de coupure(s), la probabilité de cross-over, mutation etc...

BIBLIOGRAPHIE

[1] A. Gupta, "Formal Hardware Verification Methods: A Survey", Formal Methods in System Design, An International Journal, V1: 151-238 (1992).

[2] Formal Methods Education Resources "http://www.cs.indiana.edu/formal-methods-education/"

[3] System Science, Inc. (Synopsys, Inc.), "The VERA verification system," http:// www.systems.com/products/vera.

[4] Verisity Design, Inc., "Spec-man: Spec-based approach to automate functional verification," http://www.verisity.com.

[5] 0-In Design Automation, Inc., "Whitebox verification," http://www.0-in.com/tools.

[6] TransEDA, Inc., "VeriSure- Verilog code coverage tool," http://www.transeda.com/ products.

[7] CheckOff User Guide, Siemens Nixdorf Informations Systemen AG & Abstract Hardware Limited, January, 1996.

[8] K. L. McMillan, "Symbolic model checking - an approach to the state explosion problem", Ph.D. thesis, SCS, Carnegie Mellon University, 1992. (See also Cadence Design Systems, Inc. www pages.)

[9] FormalCheck User's Guide. Bell labs Design Automation, Lucent Technologies, V1.1, 1997.

[10] 0-In Design Automation, Inc., "0-In search," http://www.0-in.com/tools.

[11] Lambda user's manual, Abstract Hardware limited, 1995.

[12] C. Darwin, "On the Origin of Species". John Murray, London. 1859.

[13] J. D. Watson, F. H. C. Crick, "A structure for Deoxyribose Nucleic Acid", Nature, International weekly journal of science, England, Vol 171, 737 (April 1953)

[14] R. Dumeur, "Synthèse de comportements animaux individuels et collectifs par Algorithmes Génétiques", Département Informatique, Institut d'Intelligence Artificielle, Université de Paris-8. 1995.

[15] J. H. Holland, "Adaptation in Natural and Artificial Systems", University of Michigan Press, Ann Arbor. 1975.

[16] D. E. Goldberg, "Genetic Algorithms in Search Optimization and Machine Learning", Addison-Wesley, 1989.

[17] A. E. Eiben, P-E. Raué, Z. Ruttkay, "Genetic algorithms with multi-parent recombination". Parallel Problem solving from nature - 3, pages 78-87, 1994.

[18] T. Back, H.-P. Schwefel, and R. Manner, "An overview of evolutionnary algorithms for parameter optimization", Journal of Evolutionary algorithms, 1:1-23, 1993.

[19] J. D. Schaffer, A. Morishima, "An adaptive crossover distribution mechanism for genetic algorithms", Proceedings of the Second Internal Conference on Genetic Algorithms, pages 36--40, Hilsdale, New Jersey, Hove and London. Lawrence Erlbaum Associates, Publishers. 1987.

[20] S. J. Louis, G. J. E. Rawlins, "Designer genetic algorithm", Proceedings of the Fourth International Conference on Genetic Algorithms, pages 53--60. University of California, San Diego, Morgan Kaufmann Publishers. 1991.

[21] A. E. Eiben, C.H.M. van Kemenade, J. N. Kok, "Orgy in the computer: Multi-parent reproduction in genetic algorithms", Third European Conference on Artificial Life, 1995.

[22] D. Whitley, "The genitor algorithm and selection pressure", Proceedings of the Third International Conference on Genetic Algorithms, pages 116--121. Phillips Laboratories, Morgan Kaufmann Publishers, Inc. 1989.

[23] G. Syswerda, "A study of reproduction in generational and steady-state genetic algorithms", Foundations of Genetic Algorithms, pages 94--101. Morgan Kauffmann Publishers. 1991.

[24] B. Manderick, P. Spiessens, "Fine grained parallel genetic algorithms", Proceedings of the Third International Conference on Genetic Algorithms, pages 428--433. Phillips Laboratories, Morgan Kaufmann Publishers, Inc. 1989.

[25] P. Spiessens, B. Manderick, "A massively parallel genetic algorithm: Implementation and first analysis", Proceedings of the Fourth International Conference on Genetic Algorithms, pages 279--286. University of California, San Diego, Morgan Kaufmann Publishers. 1991.

[26] H. Mühlenbein, M. Schomisch, J. Born, "The parallel genetic algorithm as function optimizer", Proceedings of the Fourth International Conference on Genetic Algorithms, pages 271--278. University of California, San Diego, Morgan Kaufmann Publishers. 1991.

[27] D. Hillis, "The Connection Machine", The MIT Press, Cambridge, Massachussets. 1986.

[28] M. Baldi, F. Corno, M. Rebaudengo, P. Prinetto, M. Sonza Reorda, G. Squillero, "Verification of Correctness and Performance of Network Protocols via Simulation-Based Techniques", Conference on Correct Hardware Design and Verification Methods (CHARME'97), Montreal, Canada, October 1997.

[29] E. Alba, J. M. Troya, "Genetic Algorithms for Protocol Validation", Proceedings of the 4th conference on Parallel Problem Solving (PPSN IV), Berlin, Germany, September 1996.

[30] E. M. Rudnick, G.S. Greenstein, "A Genetic Algorithm Framework for Test Generation", IEEE Transaction on Computer-Aided Design of Integrated Circuits and System. Vol. 1, No. 9. September 1997.

[31] F. Corno, P. Prinetto, M. Rebaudengo, M. Sonza Reorda. "GATTO: A Genetic Algorithm for Automatic Test Generation for Large Synchronous Sequential Circuits", IEEE Transaction on Computer-Aided Design Networking, Vol. CAD-15, No. 8, August 1996.

[32] Y. Hoskote, T. Kam, P. Ho, X. Zhao, "Coverage Estimation for Symbolic Model Checking", Proceeding of the 36th Design Automation Conference (DAC'99), Louisianna, NO, USA, July 1999.

[33] E. M. Clarke, E. A. Emerson, A. P. Sistla, "Automatic verification of finite state concurent systems using temporal logic specifications", ACM transactions on Programming languages and Systems, 8(2):244-263 (April 1986).

[34] E. A. Emerson, C. L. Lei, "Modalities for model checking: Branching time strikes back", Proceedings of the Twelfth Annual ACM Symposium on Principles of Programming Language, ACM, New York, 1985, pp. 84-96.

[35] A. Silburt, "Invited Lecture: ASIC/System Hardware Verification at Nortel: A View from the Trenches", Proceeding of the 9th IFIP WG10.5 Advanced Research Working Conference on Correct Hardware Design and Verification Methods (CHARME'97), Montreal, October,1997.

[36] A. J. Camilleri, "A role for Theorem Proving in Multi-processor Design", Proceeding of the 10th International Conference on Computer Aided Verification (CAV'98),Vancouver, June/July 1998.

[37] D. L. Dill, "What's Between Simulation and Formal Verification", Proceeding of the 35th Design Automation Conference (DAC'98), San Francisco, CA, USA, June 1998.

[38] Lucent Technologies, "FormalCheck model checker," http:/www.bell-labs.com/org/blda/product.formal.html.

[39] P. Bratley, B.L. Fox, L.E. Schrage, "A Guide to Simulation", 2nd Edition, Springer-Verlag, New York, 1986.

[40] L. Devroye, "Non-Uniform Random Variate Generation", Springer-Verlag, New York, 1986.

[41] W. Canfield, E. A. Emerson, A. Saha., "Checking Formal Specifications under Simulation", International Conference on Computer Design (ICCD'1997), Austin, Texas, October 1997.