Cadre de la thèse Optimisations arithmétiques

Cette thèse s'inscrit dans la continuité de la thèse de Yannick Dumonteix [Dum01] soutenue en octobre 2001 : "Optimisation de chemins de données arithmétiques par l'utilisation de plusieurs systèmes de numération". Il a en effet étudié l'impact de l'introduction de l'arithmétique redondante dans le flot de conception, étude qui lui a permis de formaliser une méthode complète d'outils d'optimisation utilisant l'arithmétique redondante.

Il a tout d'abord étudié l'usage conjoint de l'arithmétique redondante et de l'arithmétique classique. Il a pour cela défini une nouvelle arithmétique qu'il a appelée *arithmétique mixte*, étant la combinaison de ces deux représentations. Dans le but de montrer l'intérêt de cette nouvelle arithmétique, il a présenté les architectures des différents opérateurs en faisant une étude comparative entre les performances intrinsèques des opérateurs redondants et mixtes et celles des opérateurs classiques.

Il a ensuite déterminé l'impact de l'utilisation de ces nouvelles architectures dans la conception de circuits en étudiant les enchaînements d'opérateurs. Cette étude lui a permis de proposer des règles d'optimisations automatisables consistant à remplacer un enchaînement d'opérateurs utilisant l'arithmétique classique par des opérateurs utilisant l'arithmétique redondante. L'utilisation de ces règles permet d'optimiser automatiquement les chemins de données arithmétiques.

Son travail s'est concrétisé par la spécification d'une *méthodologie d'aide à la conception de chemins de données*. Dans cette spécification il a tout d'abord explicité un cadre de développement permettant la mise en œuvre automatisée des règles d'optimisation présentées, et a défini ensuite ce qu'il a appelé un *proto-langage*.

Dans un premier temps, nous faisons une synthèse de la méthode d'optimisation formalisée par Yannick Dumonteix. Nous présentons les différentes étapes d'optimisation, étapes qui ont servi de base à nos travaux. Nous présentons également l'environnement de conception, avec entre autres les fonctionnalités de son proto-langage, correspondant principalement aux contraintes dues à l'arithmétique redondante.

Dans un second temps, nous exposons notre problématique. Basés sur les conclusions de Yannick Dumonteix, nos travaux se résument à l'utilisation de façon automatique des règles qu'il a présenté ainsi que la mise en place du proto-langage qu'il a défini.

La méthodologie d'optimisation présentée par Yannick Dumonteix comprend deux phases distinctes :

- l'optimisation des équations arithmétiques et logiques, composée de plusieurs étapes que nous allons détailler,
- la redistribution des parties arithmétiques et logiques.

1.1.1 Phase d'initialisation

Passage tout en redondant

L'utilisation de l'arithmétique redondante consiste à instancier des opérateurs redondants à la place des opérateurs classiques d'un chemin de données arithmétique, de telle façon que :

- son interface soit inchangée,
- son comportement soit inchangé,
- son architecture soit modifiée de façon à exploiter au mieux les opérateurs redondants.

Cette méthode d'initialisation d'un chemin de données arithmétique est présentée comme apportant un gain toujours positif grâce aux meilleures performances intrinsèques des opérateurs redondants. Nous reviendrons en détails sur les performances des différents opérateurs arithmétiques dans le chapitre suivant.

Dans la suite de ce manuscrit, les signaux non redondants (noté *NR*) seront représentés par une simple flèche et les signaux redondants (noté R) par une double flèche (en effet, les nombres redondants sont codés sur deux termes). Les opérateurs redondants (opération entre deux nombres redondants) et mixtes (opération entre un nombre redondant et un nombre non redondant) seront donc représentés avec des signaux redondants (doubles flèches) pour leurs entrées/sorties redondantes et des signaux non redondants (simples flèches) pour leurs entrées/sorties non redondantes.

Un code des couleurs sera également adopté de façon à rendre plus lisibles les opérateurs utilisés : cyan pour les opérateurs classiques, magenta pour les opérateurs redondants et mixtes, et blanc pour les convertisseurs permettant de passer d'une représentation à l'autre.

La Figure 1.1 présente les différentes conventions utilisées.

Considérons l'exemple de la Figure 1.2. Notre postulat de base est que les additionneurs redondants sont plus petits et plus rapides que les additionneurs classiques. De plus, comme nous le verrons en détails dans le Chapitre 2, un nombre redondant codé en représentation Carry-Save est la somme de deux éléments, on parle alors de somme non encore effectuée ; cela permet d'effectuer la somme de quatre termes avec un seul additionneur redondant plutôt qu'avec trois additionneurs classiques.

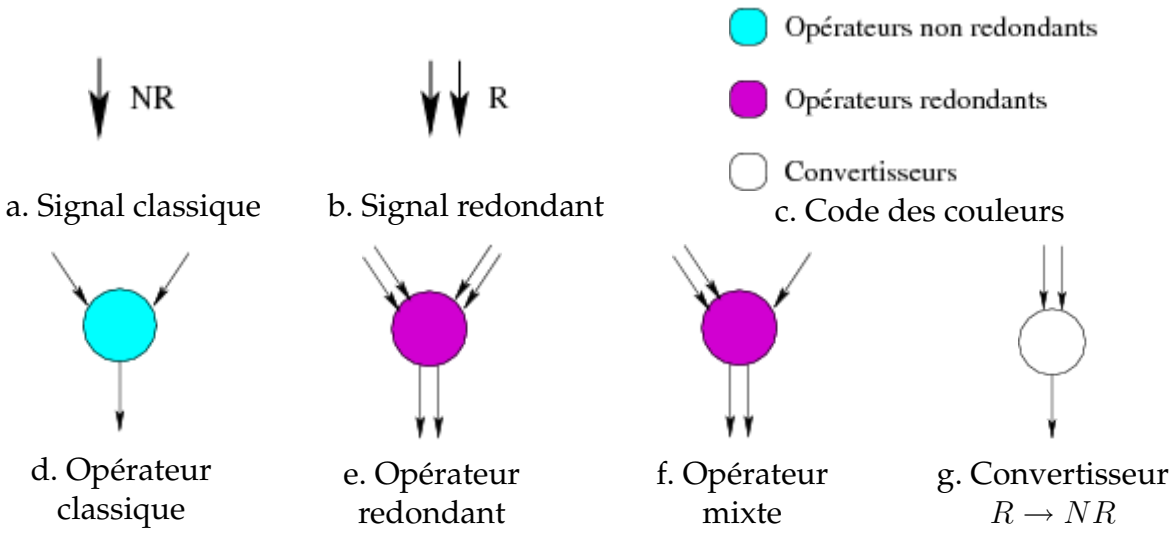


FIG. 1.1 – Légende

On déduit aisément que le *passage tout en redondant* (cf. Figure 1.2.b) génère une architecture plus petite et plus rapide que celle en arithmétique classique (cf. Figure 1.2.a).

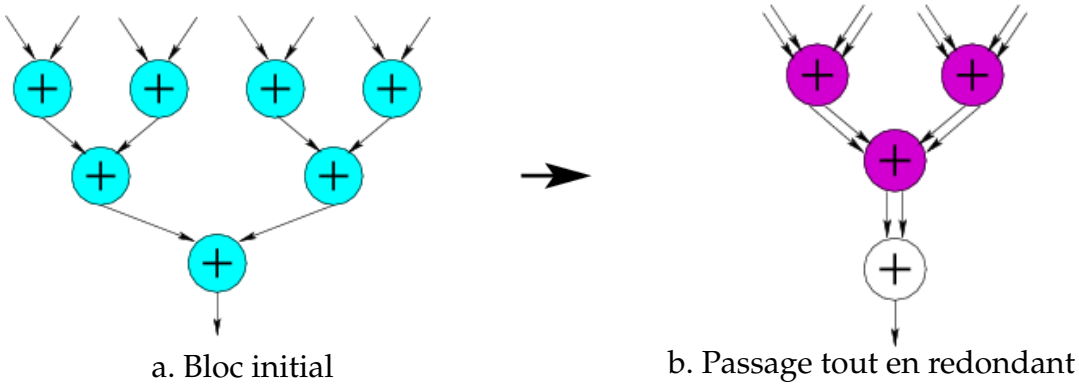


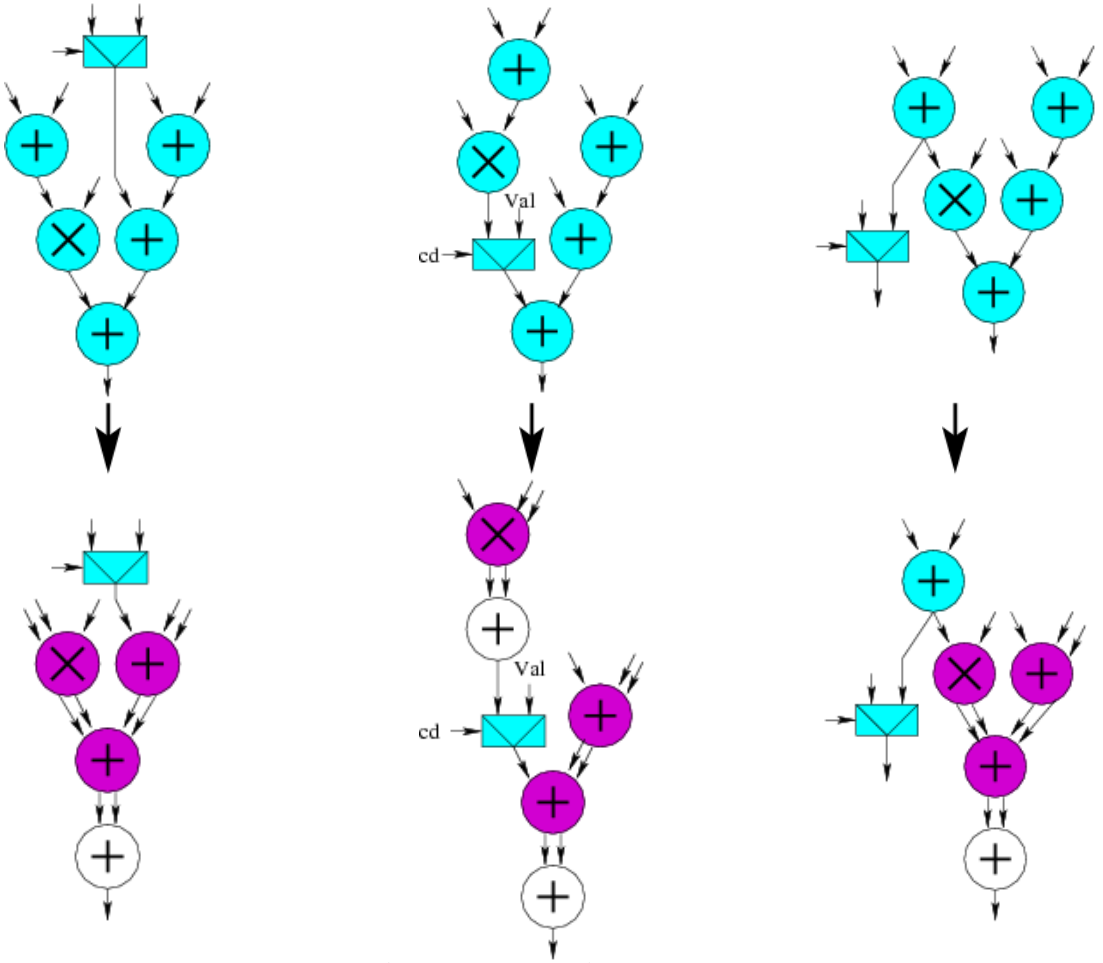
FIG. 1.2 – Phase d'initialisation

Passage en redondant dès que possible

L'exemple présenté précédemment est un cas *idéal* se réduisant uniquement à une succession d'opérateurs arithmétiques. Un chemin de données arithmétique ne se réduit cependant pas à cela, il faut également prendre en compte les opérateurs non arithmétiques, tels que les multiplexeurs et les opérateurs booléens. Ces opérateurs doivent garder les signaux à leur interface en non redondant, il est donc possible qu'ils *compromettent* le passage en redondant en obligeant l'utilisation d'opérateurs classiques. C'est pourquoi il ne faut pas parler de *passage tout en redondant* mais de *passage en redondant dès que possible*.

Trois exemples sont présentés dans la Figure 1.3. Ces exemples représentent les trois cas d'enchaînement qui peuvent survenir entre un opérateur arithmétique et un opérateur non arithmétique :

- Figure 1.3.a : le bloc non arithmétique n'est pas situé *à l'intérieur* de la chaîne arithmétique, il n'a donc pas d'influence sur les optimisations possibles,
- Figure1.3.b : le bloc non arithmétique est à l'intérieur de la chaîne arithmétique, les signaux à son interface doivent donc rester en représentation non redondante, on parle alors de limitation de type opérateur,
- Figure 1.3.c : la sortie d'un opérateur arithmétique est connecté à la fois à un bloc arithmétique et au bloc non arithmétique, cette seconde connexion empêche de passer ce signal en représentation redondante, on parle alors de limitation de type donnée.



a. Bloc sans influence

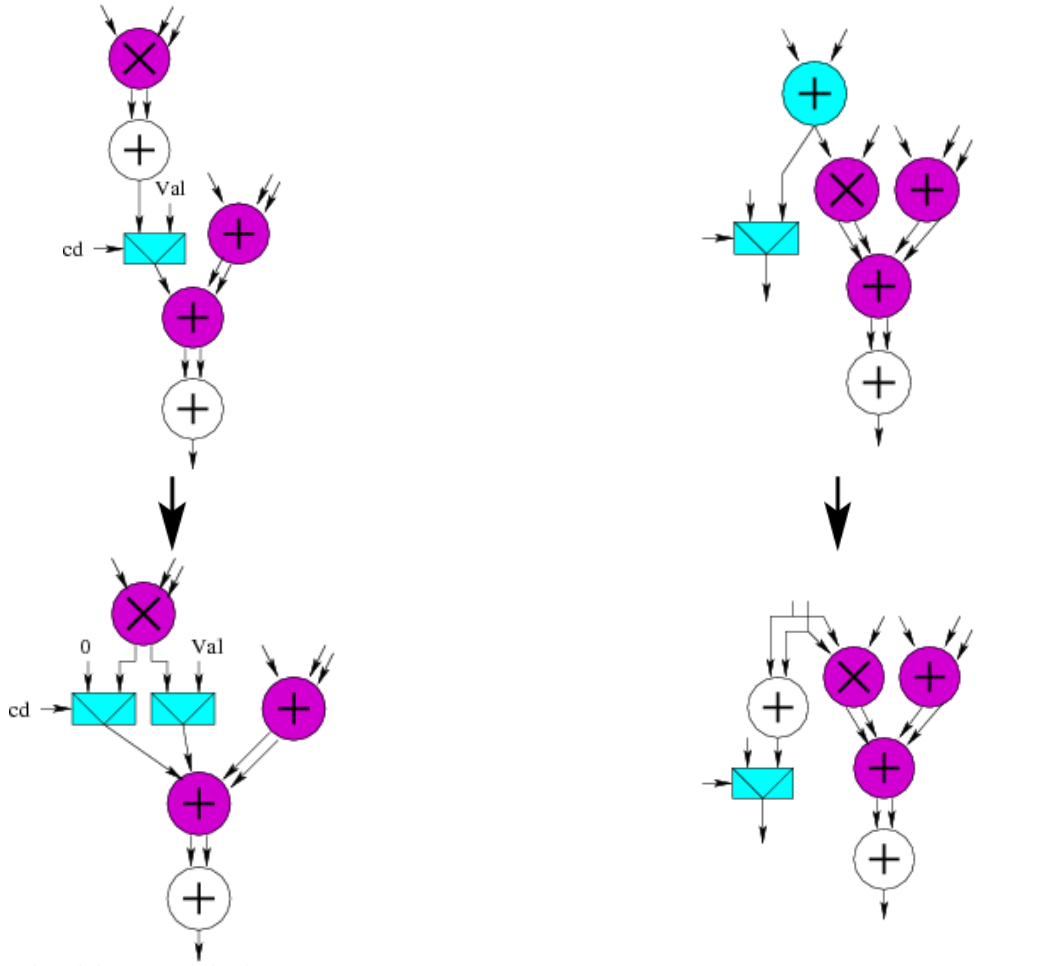
b. Limitation de type opérateur

c. Limitation de type donnée

FIG. 1.3 – Impact des blocs non redondants

De façon à pouvoir utiliser quand même des opérateurs redondants et ainsi améliorer les performances du chemin de données arithmétique, des solutions sont proposées pour les deux cas dans lesquels le bloc non arithmétique empêche le passage en redondant comme montré dans la Figure 1.4 :

- dans le cas de la limitation de type opérateur, il est possible de dédoubler l'opérateur de façon à relier chacun des deux opérateurs à un des termes du nombre redondant comme montré dans la Figure 1.4.a,
- dans le cas de la limitation de type données, il est possible de dédoubler la donnée, de ce fait, on relie la donnée en redondant au bloc arithmétique, et on relie la donnée en non redondant au bloc non arithmétique comme montré dans la Figure 1.4.b.



a. Dédoublement de l'opérateur b. Dédoublement de la donnée

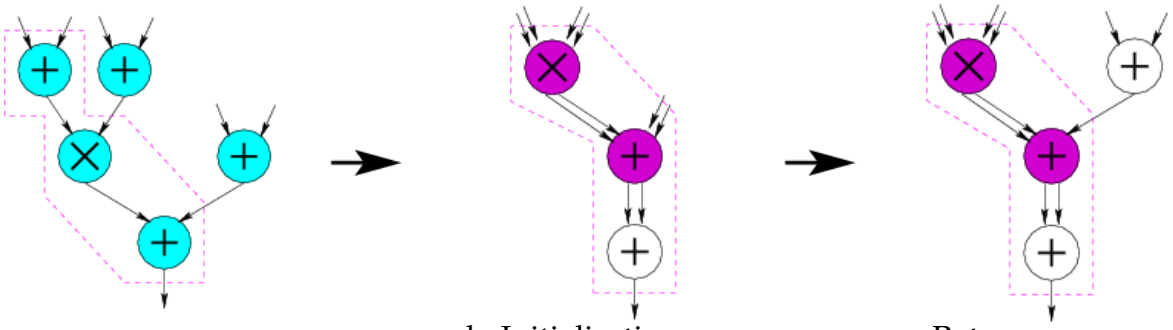
FIG. 1.4 – Dédoublement

Conditionnement par l'analyse

Les optimisations présentées précédemment apportent un gain substantiel dû aux bonnes performances intrinsèques des opérateurs redondants. Cependant, ce gain n'est pas forcément optimal pour le délai car ces optimisations ne prennent pas en compte la dimension temporelle du chaînage des opérateurs.

En effet, dans certains cas, il est préférable de ne pas transformer un opérateur en sa version redondante pour optimiser la chaîne longue. Plus précisément, l'objectif est d'exploiter les déséquilibres de chemins de façon à réduire le nombre d'entrées redondantes des opérateurs se situant dans la chaîne longue afin d'utiliser des opérateurs mixtes. Cela permet de cette façon de minimiser la chaîne longue car les opérateurs mixtes sont plus petits et plus rapides que leur équivalent redondant. Le principe est donc de partir de la version totalement redondante d'un chemin de données et d'insérer à bon escient des convertisseurs de notation redondante vers notation classique (noté $R \rightarrow NR$) permettant ainsi l'utilisation des opérateurs mixtes.

Dans l'exemple présenté dans la Figure 1.5.a, la chaîne longue passe par le chemin de gauche. Les chemins entre les deux entrées de l'opérateur de sortie sont donc déséquilibrés, l'un étant plus long que l'autre. L'idée est que l'insertion d'un convertisseur $R \rightarrow NR$ ne modifie pas le fait que la chaîne longue passe par le chemin de gauche (le chemin de droite est toujours plus court), elle permet par contre de transformer l'opérateur redondant (cf. Figure 1.5.b) en un opérateur mixte plus rapide (cf. Figure 1.5.c).



a. Bloc

b. Initialisation en redondant

c. Retour en non redondant

FIG. 1.5 – Conditionnement par l'analyse

Remarques :

1. Cette technique repose sur l'analyse des temps de propagation et doit être réitérée après chaque modification de l'architecture pour converger vers la solution optimale ; le coût potentiel d'un tel mécanisme paraît donc assez élevé.
2. Cette technique est à effectuer uniquement si l'optimisation voulue est du point de vue du délai ; en effet, elle améliore le délai au détriment de la surface, par rapport à l'optimisation tout en redondant.

1.1.2 Opérateur de somme

Une fois le chemin de données obtenu avec des opérateurs redondants, des optimisations sont encore possibles : l'étude des enchaînements d'opérateurs permet d'extraire de nouvelles règles d'optimisations.

A la base de ces règles se trouve l'opérateur de somme qu'est l'arbre de Wallace [Mul89]. Ces règles ne font donc pas parti à proprement parler des optimisations liées à l'arithmétique redondante, mais leur sont connexes.

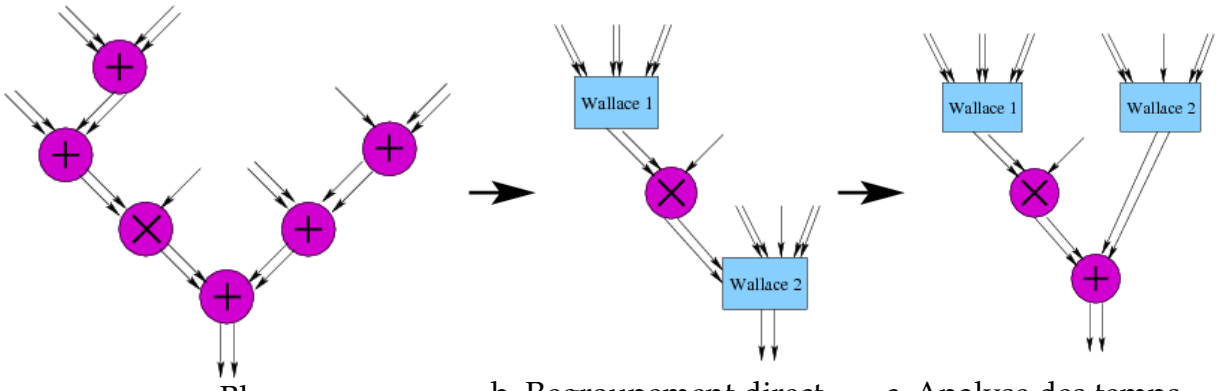
Regroupement

Il s'agit du regroupement d'une suite d'opérateurs effectuant la même opération au sein d'un unique opérateur.

La Figure 1.6 présente un exemple d'un tel regroupement dans lequel une suite d'additionneurs est regroupée au sein d'un arbre de Wallace.

Cependant, l'architecture présentée dans l'exemple est une architecture typique menant à une contre-performance : le principe est le même que le *passage tout en redondant* nécessitant un retour en non redondant de façon à être optimal. En effet, en effectuant un regroupement de tous les additionneurs en un seul abre de Wallace, le temps de propagation subit un retard (cf. Figure 1.6.b) puisque le temps de propagation d'un arbre de Wallace est supérieur à celui d'un additionneur redondant. Ainsi il serait plus judicieux de garder l'additionneur redondant de sortie et d'utiliser un arbre de Wallace en parallèle avec la multiplication, plutôt que d'utiliser un arbre de Wallace après la multiplication. Une analyse de temps s'impose donc afin de *découper* les regroupements (cf. Figure 1.6.c).

La méthode à mettre en place est une nouvelle fois d'effectuer tous les regroupement possibles dans un premier temps puis de les *découper* en fonction de l'analyse des temps.



a. Bloc b. Regroupement direct c. Analyse des temps

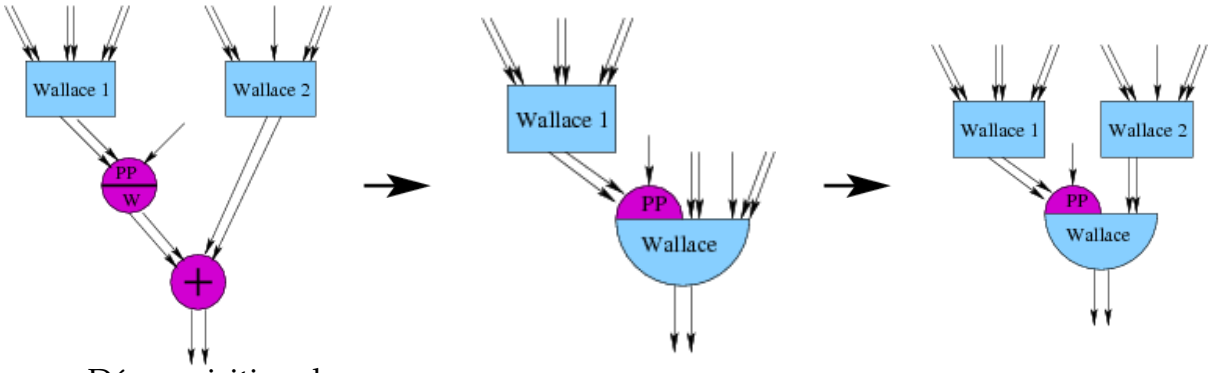
FIG. 1.6 – Regroupement

Fusion

Il s'agit de l'absorption d'un opérateur par un autre de nature différente.

La Figure 1.7 présente l'exemple d'une fusion, dans laquelle un multiplieur est découplé en deux parties (cf. Figure 1.7.a) de façon à pouvoir effectuer un regroupement d'additionneurs avec l'arbre de Wallace interne du multiplieur.

Dans cet exemple, on soulève le même problème que précédemment : effectuer des fusions sans analyse des temps peut mener à des contre-performances (cf. Figure 1.7.b). La solution à adopter est donc la même : effectuer une analyse temporelle de façon à *découper* les fusions (cf. Figure 1.7.c).



a. Décompisition du multiplieur : produits partiels + somme

b. Fusion directe

c. Analyse des temps

FIG. 1.7 – Fusion

Synthèse

Comme cela a été montré dans les deux exemples précédents, le gain en temps n'est pas toujours positif en ce qui concerne les regroupements et les fusions. En effet, ces optimisations dépendent très fortement du contexte.

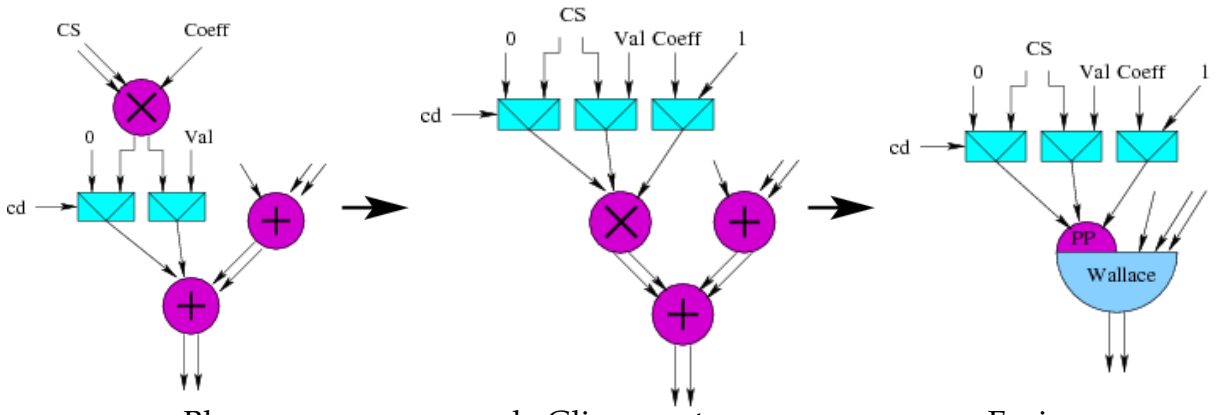
Il faut donc mettre en place une méthodologie d'optimisation pouvant calculer les temps de propagation des opérateurs de façon à être capable de corriger les éventuels effets indésirables.

1.1.3 Glissement

Le glissement est une redistribution des ressources logiques de façon à dégager des portions arithmétiques plus importantes pour permettre davantage d'optimisations arithmétiques.

Là encore, ce n'est pas une technique propre à l'utilisation de l'arithmétique redondante : elle a pour but de *préparer* les circuits en vue d'une optimisation arithmétique.

Dans l'exemple de la Figure 1.8.a, nous pouvons voir que les multiplexeurs à l'intérieur de la chaîne arithmétique ne permettent pas de fusionner les deux additionneurs avec le multiplieur. Le déplacement des deux multiplexeurs se traduit par l'allongement du chemin de données purement arithmétique (cf. Figure 1.8.b), ce qui permet d'effectuer la fusion (cf. Figure 1.8.c).



a. Bloc b. Glissement c. Fusion

FIG. 1.8 – Glissement

1.1.4 Circuits séquentiels

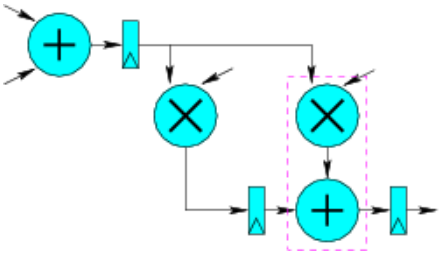
Dans les sections précédentes nous nous sommes restreint aux circuits combinatoires. Yannick Dumonteix a également fait l'étude de l'intérêt de sa méthode en ce qui concerne les circuits séquentiels.

En théorie, les registres peuvent être doublés comme n'importe quel bloc non arithmétique en utilisant la technique de dédoublement présentée dans la Figure 1.4.a. Cependant, leur qualité de *barrière temporelle* fait que cette action peut être source de gain dans certains cas et de contre performance dans d'autres, comme montré dans l'exemple de la Figure 1.9.

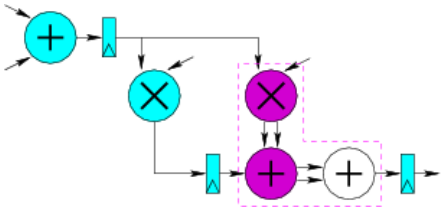
Un bloc arithmétique séquentiel est montré dans la Figure 1.9.a, et l'optimisation de ses enchaînements combinatoires dans la Figure 1.9.b. Deux sortes d'optimisation à travers les registres sont montrées dans les Figures 1.9.c et 1.9.d. Elles se différencient par la place choisie pour le convertisseur final :

- dans la première, le temps de propagation est amélioré car la traversée des registres par des nombres redondants a permis la disparition de conversions $R \rightarrow NR$,
- la seconde amène cependant à une contre performance, en effet dans ce deuxième cas :

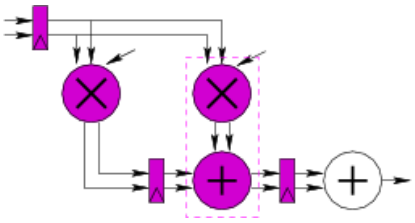
  1. la traversée du registre noté 1 par un nombre redondant a impliqué l'augmentation du délai de l'arbre combinatoire rattaché à la sortie de ce registre



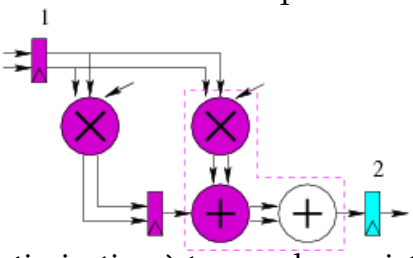
a. Bloc



a. Optimisation des enchaînement combinatoires uniquement



b. Optimisation à travers les registres : Sortie combinatoire



c. Optimisation à travers les registres : Sortie sur registre

FIG. 1.9 – Circuit séquentiel

(les opérateurs de cet arbre voient leur nombre d'entrées redondantes augmenter),

2. en parallèle, la traversée du registre noté 2 par un nombre classique a elle aussi impliqué l'augmentation du délai de l'arbre combinatoire rattaché à l'entrée de ce registre (donc le même que précédemment) de part la nécessité de conserver la conversion $R \rightarrow NR$.

La conclusion donnée est donc que le passage des notations redondantes à travers les registres doit être conditionnée par une analyse des temps de propagation.

1.1.5 Méthodologie générale

Les différentes phases composant la méthodologie d'optimisation de Yannick Dumonteix viennent d'être présentées :

- **passage en redondant partout où c'est possible** : choix des architectures en utilisant autant d'opérateurs redondants / mixtes que possible,
- **glissements** : redistribution des données afin d'avoir des chaînes arithmétiques à optimiser plus longues,
- **regroupements** et **fusions** : utilisation d'arbres de Wallace,
- **retours en notation non redondante** : analyse des temps de propagation pour introduire des retours en notation non redondante de façon à améliorer la chaîne critique.

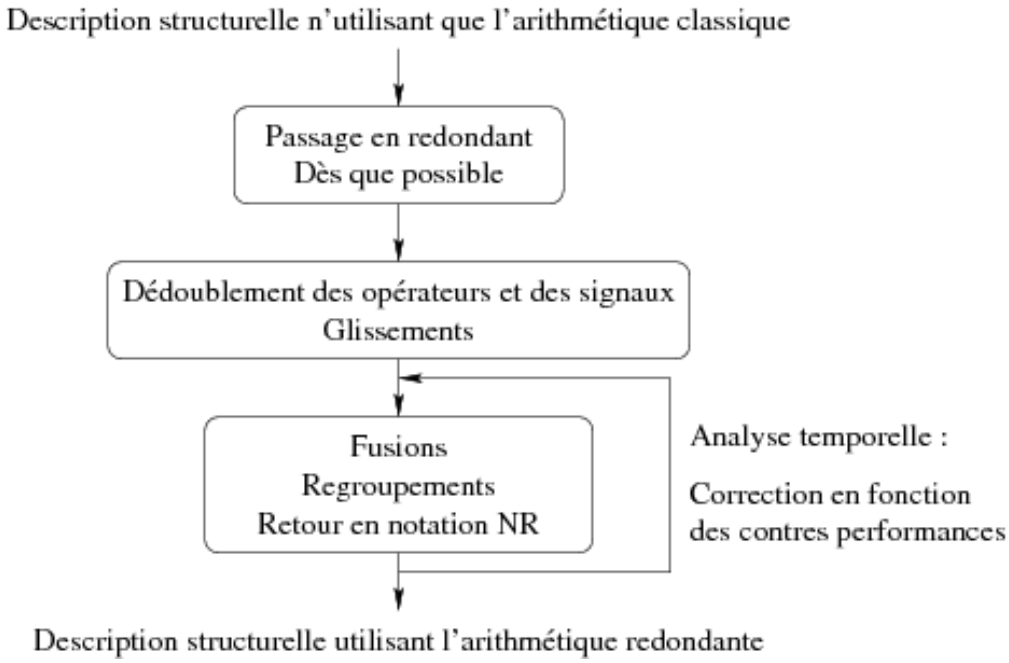


FIG. 1.10 – Etapes de la méthode

Ces phases sont encapsulées dans une méthodologie générale, comme montré dans la Figure 1.10. Après ces différentes étapes, l'architecture des opérateurs ainsi que les interconnexions dans le circuit ont été spécifiées puis éventuellement modifiées de façon à améliorer les performances.

1.2 Environnement de conception

La méthodologie de conception présentée par Yannick Dumonteix va au-delà de la mise en œuvre des différentes phases d'optimisations arithmétiques que nous venons de décrire. Elle aborde également divers autres aspects tels la conception des circuits (langage de description) ou leur portabilité (indépendance entre la description et la technologie cible), toujours dans l'optique de faciliter le but premier des concepteurs : la rapidité de conception.

1.2.1 Langage *mou*

Du point de vue de la description des circuits, le principe de la méthodologie présentée est que, dans la description initiale du circuit à optimiser, les différents opérateurs ne sont pas *complètement spécifiés* : les concepteurs fournissent une description la moins définie possible, i.e. une description structurelle dans laquelle les opérateurs ne sont connus que par leurs interconnexions et leur fonctionnalité ; le terme employé est *description molle*.

La création des opérateurs peut alors se faire avec une forme implicite ($a + b$ par exemple pour l'opération d'addition), plutôt qu'avec une forme explicite comme cela est fait couramment dans les langages de description de vues structurelles (instanciation de l'opérateur avec architecture choisie et interconnexions).

D'un point de vue arithmétique, les signaux doivent contenir l'information de la différenciation entre les différentes représentations arithmétiques des nombres.

1.2.2 Portabilité

Après les étapes d'optimisation, la description obtenue est une description bas niveau la plus précise possible, i.e. une liste de blocs élémentaires avec leur architecture et leurs interconnexions, dans une technologie cible donnée.

Cependant, l'idée de portabilité est elle aussi très importante dans un but de rapidité de conception, par réutilisation de l'existant. Une description intermédiaire prend alors tout son sens : après optimisation, mais avant projection vers une technologie cible. On parle alors de description en portes fonctionnelles ou virtuelles plutôt que réelles.

De cette façon, le concepteur a à disposition une version optimisée de son circuit indépendante de la technologie cible. Il peut ainsi faire suivre les évolutions technologiques au circuit, et également le changer de technologie, cela sans coût additif.

Remarques :

1. Notons que la version optimisée d'un circuit est indépendante de la technologie cible uniquement si l'algorithme d'optimisation utilisé est lui aussi indépendant de cette technologie.

   Comme nous le verrons, un de nos algorithmes dépend de la technologie cible, du fait qu'il prend en compte les temps de propagation des cellules. On obtient alors bien un circuit en portes virtuelles, mais l'optimisation effectuée est optimale pour une technologie en particulier, rien ne garantit qu'elle le soit pour une technologie différente.

   Il est donc nécessaire d'adapter l'algorithme suivant la technologie utilisée, i.e. les temps de propagation des cellules virtuelles sur lesquelles il se base.

2. Il est suggéré que cette description avant projection soit faite dans le langage d'entrée, de façon à pouvoir, entre autres, réutiliser les développements antérieurs.

   Une fois la projection vers une technologie cible effectuée, il est par contre conseillé que la description soit dans un langage connu de façon à être facilement interfaçable avec le reste du flot de conception.

1.3 Problématique

L'objectif de cette thèse est de proposer un outil de conception de circuits permettant l'optimisation des chemins de données arithmétiques grâce à l'utilisation de l'arithmétique redondante tout en facilitant le travail des concepteurs.

Pour cela, nous nous appuierons sur les travaux de Yannick Dumonteix qui a été l'un des premiers à étudier l'impact de l'introduction de l'arithmétique redondante dans le flot de conception *VLSI*. Il a traité pour cela la problématique concernant la mise en place et l'évaluation de l'arithmétique redondante au côté de l'arithmétique classique, définissant pour cela l'arithmétique mixte.

Cette étude a été étoffée par la conception *à la main* de divers circuits arithmétiques utilisant cette arithmétique tels un opérateur de transformée en cosinus discrète [DCM00] et un opérateur de calcul de distance [DBM01]. Ces diverses études ont abouti à de meilleures performances des architectures mises en œuvre.

Suite à ces résultats concluants, Yannick Dumonteix a défini diverses optimisations arithmétiques applicables de façon automatique. Il a alors encapsulé ces optimisations dans une méthodologie d'optimisation complète.

Cependant, appliquer *à la main* la méthodologie qu'il a présentée n'est pas chose aisée pour des concepteurs de circuits n'ayant pas forcément le savoir-faire arithmétique nécessaire. Cela s'avère par ailleurs difficile voire impossible sur de gros circuits. Il en découle naturellement l'idée d'encapsuler cette méthodologie dans un outil qui la mettra en œuvre de manière automatique.

Une telle automatisation a pour but de rendre l'utilisation de l'arithmétique redondante beaucoup plus accessible, permettant aux concepteurs n'ayant pas le savoir-faire requis de pouvoir néanmoins tirer partie de cette arithmétique et d'améliorer de façon significative les performances de leurs circuits.

Notre problématique se focalise donc sur la mise en place de l'automatisation de la méthodologie d'optimisation arithmétique à travers deux étapes fondamentales :

1. L'optimisation arithmétique à proprement parler : différents types d'algorithmes sont utilisables de façon à mettre en pratique automatiquement les différentes optimisations proposées par Yannick Dumonteix. Nous voulons donc définir le meilleur procédé pour encapsuler ces optimisations.
2. L'environnement de conception : nous voulons *faciliter le travail* des concepteurs en caractérisant précisément leurs besoins du point de vue du langage d'entrée, des services à fournir, etc. Ceci sous-entend fournir un langage complet, clair et intuitif. Ce langage doit également permettre de définir un chemin de données par une description simplifiée, i.e. considérer les opérateurs élémentaires (arithmétiques, booléens) comme de simples mnémoniques. Il doit enfin répondre aux besoins de l'arithmétique i.e. permettre la spécification de la représentation des signaux.

1.4 Conclusion

Dans ce chapitre nous avons présenté les différentes optimisations arithmétiques utilisant l'arithmétique redondante, introduites par Yannick Dumonteix dans sa thèse, ainsi que l'environnement de conception associé. Suite à cela, nous avons exposé notre problématique.

Nous allons maintenant présenter plus en détail les caractéristiques de l'arithmétique redondante de façon à démontrer l'intérêt de notre démarche.