Titre: Title:	Méthode de détermination automatique de la taille des chemins de données
Auteur: Author:	Marc-André Cantin
Date:	2005
Type:	Mémoire ou thèse / Dissertation or Thesis
Référence: Citation:	Cantin, M.-A. (2005). Méthode de détermination automatique de la taille des chemins de données [Ph.D. thesis, École Polytechnique de Montréal]. PolyPublie. https://publications.polymtl.ca/7547/

UNIVERSITÉ DE MONTRÉAL

MÉTHODE DE DÉTERMINATION AUTOMATIQUE DE LA TAILLE DES CHEMINS DE DONNÉES

MARC-ANDRÉ CANTIN
DÉPARTEMENT DE GÉNIE ÉLECTRIQUE
ÉCOLE POLYTECHNIQUE DE MONTRÉAL

THÈSE PRÉSENTÉE EN VUE DE L'OBTENTION
DU DIPLÔME DE PHILOSOPHIÆ DOCTOR
(GÉNIE ÉLECTRIQUE)
JUIN 2005

Your file Votre référence
ISBN: 978-0-494-16984-1
Our file Notre référence
ISBN: 978-0-494-16984-1

UNIVERSITÉ DE MONTRÉAL

ÉCOLE POLYTECHNIQUE DE MONTRÉAL

Cette thèse intitulée:

MÉTHODE DE DÉTERMINATION AUTOMATIQUE DE LA TAILLE DES CHEMINS DE DONNÉES

présentée par: CANTIN Marc-André
en vue de l'obtention du diplôme de: Philosophiæ Doctor
a été dûment acceptée par le jury d'examen constitué de:

M BOIS Guy, Ph.D., président
M. SAVARIA Yvon, Ph.D., membre et directeur de recherche
M. BLAQUIÈRE Yves, Ph.D., membre et codirecteur de recherche
M. KHOUAS Abdelhakim, Ph.D., membre
M. BELZILE Jean, Ph.D., membre

REMERCIEMENTS

Avant de commencer, je tiens personnellement à remercier mon directeur de recherche Dr Yvon Savaria de l'École Polytechnique de Montréal pour ses conseils judicieux dans mes recherches, son support financier dans ce projet, et de m'avoir permis de présenter mes recherches aux quatres coins du monde, sans oublier mon co-directeur de recherche Dr Yves Blaquière de l'Université du Québec à Montréal. Également, je désire remercier le Département de la Défense Nationale et tout spécialement Dr Pierre Lavoie, d'avoir été un instigateur de ce projet, pour les conseils techniques, l'introduction rigoureuse à l'écriture scientifique et le support financier. J'aimerais mentionner que la CMC (Canadian Microelectronic Corporation) m'a permis d'utiliser de nombreux outils de conception en microélectronique, et ce, à la fine pointe de la technologie.

Je remercie plusieurs amis et collaborateurs tel que Louis-Pierre Lafrance, Serge Catudal, Sébastien Regimbal, Ghislain Provost, Bruno Tanguay, René Taillefer et Mathieu Renaud d'avoir pris part à mes recherches et à ma vie étudiante.

Je ne peux oublier de remercier M Réjean Lepage et Mme Ghyslaine Ethier Carrier, des personnes importantes pour moi et sans aucun doute pour le GRM (Groupe de Recherche en Microélectronique), qui ont toujours eu une écoute attentive et des gestes attentionnés à mon égard.

Finalement, je remercie particulièrement ma conjointe adorée Katie Lavallée pour son soutien, son encouragement, sa patience et d'avoir cru en moi tout au long de ce long périple. À mes parents et ma famille pour leur encouragements dans cette voie, je vous dis Merci!

RÉSUMÉ

Tandis que la majorité des algorithmes d'analyse et de traitement des signaux sont développés en virgule flottante, leur implantation matérielle requiert fréquemment des opérateurs à virgule fixe afin de rencontrer les contraintes de coût et de performance. Dans le but de conserver les propriétés de l'algorithme original, d'éviter les erreurs de débordement ou de perte de précision, chaque opérande doit être représentée par un nombre de bits adéquat.

Une nouvelle méthode qui détermine automatiquement les tailles des chemins de données est présentée. Cette méthode est basée sur une nouvelle métrique qui grade toutes les combinaisons des tailles des chemins de données vers la solution optimale. Cette gradation s'effectue selon les coûts d'implantations et selon plusieurs modèles d'erreurs et contraintes de précision spécifiés par le concepteur matériel. Une procédure de recherche qui maximise cette métrique, minimise l'écart entre les modèles représentés en format virgule fixe et virgule flottante, et obtient une solution optimisée. Quatre nouvelles procédures de recherche sont proposées. Ces procédures sont comparées à celles retrouvées dans la litterature qui ont été adaptées et implantées dans la méthode. La comparaison permet de sélectionner une procédure qui est capable de trouver le plus rapidement possible une solution qui rencontre les contraintes de précision spécifiées par le concepteur et qui minimise la taille des chemins de données.

Cette méthode a été modifiée afin d'optimiser les algorithmes itératifs en déterminant le nombre de fois qu'une iteration doit être exécutée. Cette modification permet de trouver des solutions sans analyse complexe des algorithmes itératifs. Elle est applicable pour la plupart des algorithmes où il n'existe pas de modèle analytique disponible dans la litérature. Dans les cas où il existe de tels modèles, ces modèles restreignent souvent les optimisations résultantes à des solutions plus coûteuses.

ABSTRACT

Digital Signal Processing (DSP) algorithms are often expressed in 32- or 64-bit fixed- or floating-point data formats since these are available in most commercial off-the-shelf processors. Yet, in spite of very significant improvements in processor performance and power efficiency, the requirements of many real-time applications in terms of pure performance or low power operation command the use of specialized hardware. Since cost, power dissipation, and performance are highly dependent on the number of bits used to represent data, and on the use of floating-point operators, the problem of precise word length determination is important

A new automatic method for determining the word lengths in hardware data paths that considers these requirements is proposed and analyzed. The search-based method uses a C/C++ fixed-point simulation tool to model the impact of finite word lengths on overall accuracy. By computing dissimilarities between fixed-point and floating-point simulation results, a procedure searches for a combination of word lengths that meets accuracy criteria specified by the designer. This method is presented with four novel search procedures. These four automatic procedures are compared with other procedures given in the literature and adapted into the method. The comparison helps to select a procedure that finds a combination of word lengths that meets user specifications, in a small number of iterations.

This method was modified to determine, in a single optimization process, the combination of word lengths in hardware datapaths and the number of times that an iteration should be executed in iterative algorithms. Solutions are found with the modification, without the need of tedious analytic developments. As for most iterative algorithms, no analytic model is available in the literature, and even when they could be found, they often restrict the hardware datapaths to some given word length which prevents finding a solution with minimum implementation costs.

TABLE DES MATIÈRES

LISTE DES FIGURES

LISTE DES TABLEAUX

LISTE DES NOTATIONS ET DES SYMBOLES

AWLDT: Outil de détermination automatique

AWLDU: Unité de détermination automatique

C: Métrique de qualité

DFG: Graphe de flux de données

DSP: Traitement de signaux numériques

E: Nombre total de modèles d'erreur

ε_e: Modèle d'erreur

FWL: Partie Fractionnaire d'un chemin de données

H: Coût d'implantation

I: Nombre total d'opérandes

IDCT: Transformée discrète de cosinus inverse

IEEE: Institute of Electrical and Electronics Engineers

IWL: Partie entière d'un chemin de données

L: Triangle de Pascal

MWL: Taille minimun

μ: Moyenne

N: Nombre d'itérations

O_i: Opérande i

$\mathcal{Q}$: Métrique de qualité

S_e: Critère de précision

σ: Écart type

s_i: Format de la représentation

WL: Taille d'un chemin de données

$WL^{\max}$: Taille maximale d'un chemin de données

INTRODUCTION

Les algorithmes d'analyse et de traitement de signal sont souvent développés avec une représentation virgule flottante, 32 ou 64 bits, car ces représentations correspondent souvent à celles des processeurs commerciaux disponibles sur le marché. Malgré l'augmentation récente de la performance de ces processeurs, pour plusieurs applications temps réel, la demande en terme de consommation de puissance ou de performance requière souvent une implantation dans un circuit dédié. Sachant que le coût d'implantation, la consommation de puissance et la performance d'un circuit sont étroitement liés à l'utilisation des opérateurs virgule flottante et à la taille des chemins de données dans un circuit, le problème de conversion de l'algorithme d'une représentation virgule flottante à une représentation virgule fixe et le problème d'optimisation de la taille des chemins de données sont importants.

Voici quelques aspects qui démontrent l'importance d'optimiser le nombre de bits qui définissent les chemins de donnés dans un circuit.

Consommation de puissance: Pour les systèmes embarqués tel que les téléphones cellulaires, les ordinateurs portatifs, les agendas électroniques, les satellites et les balladeurs, la réduction de la consommation de puissance permet d'augmenter leur autonomie. Dans un circuit intégré, la consommation de puissance survient en partie lorsque les transistors passent d'un état ouvert à l'état fermé, ou *vice-versa*. Sachant que la minimisation du nombre de transistors dans un circuit intégré réduit la consommation de puissance, la réduction du nombre de bits qui définissent les chemins de données dans un circuit permet: de réduire le nombre de transistors, d'exécuter ces applications dans des circuits plus petits, de réduire la consommation de puissance et d'augmenter l'autonomie de ces systèmes.

Performance: Lorsque les chemins de données dans un circuit sont définis par un petit nombre de bits, le circuit produit des réponses rapides. À l'inverse, lorsque les chemins de données dans un circuit sont définis par un grand nombre de bits, le circuit produit généralement des résultats avec une grande précision. Donc lors de la conception d'un circuit, le concepteur doit normalement chercher un compromis entre la précision des résultats et la vitesse de traitement.

Coûts matériels: Lorsqu'un circuit intégré est fabriqué en plusieurs exemplaires, certains d'entre eux ne fonctionneront jamais, dû à certains facteurs tel qu'un mauvais alignement de masques, le dépôt d'impûretées, l'imperfection de la surface du silicium et certaines caractéristiques des solutions chimiques utilisées. Le rapport du nombre de circuits fonctionnels sur le nombre d'exemplaires fabriqués se nomme le rendement. Le rendement décroît avec la surface d'un circuit. Il est donc important de limiter la surface d'un circuit à la plus petite taille possible et de limiter la taille des chemins de données aux plus petits nombres de bits possibles.

Temps de conception: Les concepteurs de circuits intégrés doivent déterminer le plus rapidement possible la taille optimale en bits, pour chacun des chemins de données contenus dans l'implantation d'un algorithme de traitement de signal. Cette détermination s'effectue en considérant certaines contraintes tel que le temps de réponse du circuit, la précision des résultats, la surface maximale de silicium disponible et les critères reliés au domaine d'application.

La plupart du temps, la solution optimale n'est obtenue que par l'essai, l'analyse et la comparaison de toutes les combinaisons des tailles possibles des chemins de données. Prenons par exemple le filtre elliptique du 5^e ordre [24]. Ce filtre possède 32 chemins de données différents. Si chaque chemin de données peut-être défini avec une résolution qui varie de 1 à 32 bits, il y a $32^{32} = 1.46 \times 10^{48}$ différentes combinaisons possibles pour implanter ce filtre. Si 0.75 seconde est requis pour analyser

une combinaison par simulation, alors 3.48×10^{40} années seraient nécessaires pour analyser toutes les combinaisons possibles. Au-dessus de 5 chemins de données, l'analyse exhaustive par simulation est impraticable.

Aspect économique: L'aspect économique est l'aspect le plus important de tous les aspects mentionnés précédemment. Deux exemples sont montrés ici pour illustrer l'influence de l'aspect économique et comment il surplombe tous les autres aspects.

1. Si un processeur de 32 bits est nécessaire au lieu d'un processeur de 16 bits, alors la consommation de puissance pourrait être plus grande. Afin de préserver l'autonomie d'un système embarqué, des batteries plus coûteuses et plus durables seront nécessaires.

2. L'optimisation de la taille des chemins de données est une tâche ardue qui peut nécessiter jusqu'à 50% du temps de conception d'un circuit [32]. Un outil qui détermine automatiquement la taille des chemins de données peut réduire considérablement le temps de conception et donc le coût associé à la conception du circuit.

Pour tous ces aspects, une méthode qui détermine automatiquement les tailles minimales des chemins de données devient importante lors de l'implantation des algorithmes de traitement de signal.

Organisation de la thèse

Cet ouvrage contient 3 chapitres. Le prochain chapitre présente les approches retrouvées dans la littérature sur le domaine d'optimisation des tailles des chemins de données. Pour chacune d'entre elles, les avantages et les inconvénients y sont reportés. L'approche par simulation ressort de l'analyse comme celle étant la plus

précise, cependant les méthodes reportées dans la littérature et qui sont basées sur l'approche par simulation ne peuvent déterminer une solution en se basant simultanément sur le coût, la performance et les critères de précision. C'est pourquoi une nouvelle méthode qui détermine automatiquement les tailles des chemins de données est présentée. Cette méthode est basée sur une nouvelle métrique qui grade toutes les combinaisons des tailles des chemins de données vers la solution optimale. Cette gradation s'effectue selon les coûts d'implantations et selon plusieurs modèles d'erreurs et contraintes de précision spécifiées par le concepteur matériel. Nous proposons des procédures de recherche qui maximisent cette métrique, minimisent l'écart entre les modèles virgule fixe et virgule flottante, et obtiennent des solutions optimisées. Quatre nouvelles procédures de recherche sont proposées dans ce chapitre. Ces quatres procédures de recherche sont comparées à celles retrouvées dans la littérature qui ont été adaptées à notre méthode. La comparaison aide à sélectionner une procédure qui est capable de trouver le plus rapidement possible une solution, qui rencontre les contraintes de précision spécifiées par le concepteur et qui minimise les tailles des chemins de données.

Suite à cette comparaison il sera possible d'observer que cette méthode peut-être appliquée à une grande variété d'algorithmes de traitement de signal. Plus spécifiquement, dans le Chapitre 2, cette méthode est utilisée dans un environnement unifié qui analyse les implantations matérielles d'algorithmes de traitement d'image et de réduction de bruit. La méthode détermine le coût d'implantation matériel minimum qui rencontre certains critères de qualité dans les images produites par l'algorithme analysé, lorsque les images sources sont affectées par différents niveaux et types de bruit. Finalement, cet environnement unifié a la capacité de distribuer l'analyse sur plusieurs ordinateurs dans le but de réduire le temps de traitement inhérent à la méthode lorsque plusieurs images sont traitées.

Le Chapitre 3 présente deux extensions de la méthode. La première extension

permet d'optimiser les algorithmes à caractère itératifs en déterminant le nombre d'iterations devant être exécutées. Afin de rencontrer les critères de précision, la méthode étendue ajuste simultanément les tailles des chemins de données et le nombre de fois qu'une itération doit être effectuée avec l'objectif de minimiser les coûts d'implantation de l'algorithme. Cette méthode étendue est appliquée sur un algorithme itératif, et la méthode trouve différentes solutions selon l'architecture désirée. La deuxième extension, présentée au Chapitre 3, consiste à déterminer la valeur optimale des paramètres contenus dans un algorithme. Cette extension permet d'ajuster précisément les valeurs des paramètres lorsque l'algorithme est représenté en virgule fixe, et de déterminer automatiquement ces valeurs qui autrement sont souvent déterminées empiriquement [55][45][40].

Contributions

En résumé, les travaux de recherche réalisés dans le cadre de cette thèse ont permis d'apporter les contributions suivantes.

1. Élaboration d'une nouvelle méthode qui détermine automatiquement la résolution en bits des chemins de données sur la base de simulations en virgule fixe [14][10][11].

2. Élaboration d'une métrique capable de tenir compte simultanément de plusieurs mesures et critères de précision spécifiés par l'utilisateur et du coût d'implantation [15].

3. Proposition de 4 nouvelles procédures de recherche qui maximisent la métrique proposée et qui trouvent le plus rapidement possible des solutions optimisées [14][13][10][11].

4. Synthèse et analyse des procédures de recheche existantes dans la littérature qui ont été adaptatées et implantatées dans la méthode proposée [13][10][11].

5. Comparaison des différentes procédures, détermination des avantages et inconvénients de chacunes d'entre elles, et sélection de celles qui sont les plus aptes à trouver rapidement une solution optimisée [13][10][11].

6. Démonstration de l'utilitée de la méthode par l'application de la méthode sur des algorithmes de traitement vidéo et de réduction de bruit [9].

7. Adaptation de la méthode afin d'optimiser les algorithmes à caractère itératifs en déterminant à la fois le nombre de fois qu'une itération doit être effectuée ainsi que les tailles des chemins de données [12].

Compte tenu de l'organisation de cette thèse par article, pour laquelle chaque article comporte une revue de littérature pertinente, l'auteur a choisi de ne pas tenter de faire une revue de littérature unique et complète qui serait nécessairement redondante en regard de celle incluse dans chaque article.

CHAPITRE 1

DÉTERMINATION AUTOMATIQUE DE LA TAILLE DES CHEMINS DE DONNÉES

Au cours des dix dernières années, plusieurs techniques ont été proposées pour convertir les algorithmes d'une représentation virgule-flottante à une représentation virgule-fixe. Plusieurs de ces techniques utilisent des heuristiques et des méthodes analytiques dédiées à des applications spécifiques [7][68][42] qui ne peuvent pas être généralisées à d'autres applications.

Pour les applications générales, la conversion automatique d'algorithmes de traitement de signaux, d'une représentation virgule flottante à une représentation virgule fixe, consiste à déterminer la plage d'opération et la résolution minimale de chaque opérande O_i contenu dans l'algorithme, où chaque opérande O_i représente un chemin de données dans l'implantation matérielle, $i = 0, 1, \ldots, (I-1)$, et où I est le nombre d'opérandes que l'on désire convertir. La plage d'opération et la précision minimale de chaque opérande sont déterminées selon le nombre de bits alloués dans les parties entière (IWL) et fractionnaire (FWL). Par la suite, le nombre total de bits (WL) de chaque chemin de données peut être obtenu par l'équation suivante:

$$WL_i = IWL_i + FWL_i + s_i \tag{1.1}$$

où $s_i = 0$ lorsque les valeurs que peut prendre l'opérande O_i sont toujours positives, ou autrement $s_i = 1$.

Dans la littérature, trois approches sont proposées pour effectuer cette conversion. La première approche détermine IWL_i, FWL_i et s_i de chaque opérande O_i en propageant à travers un Graphe de Flux de Données (DFG) de l'algorithme, les valeurs maximales et minimales de chaque opérande [32][58]. Cette approche est rapide, mais elle tend à surestimer le nombre de bits nécessaires pour implanter les algorithmes de traitement de signaux. De plus, cette technique requière que le nombre de bits des entrées soit préalablement connu, et elle devient complexe à utiliser lorsque les algorithmes analysés contiennent des dépendances de données.

La seconde approche détermine le nombre de bits que doit prendre chaque chemin de données à partir d'une analyse statistique. La méthode proposée dans [25] par Cmar *et al.*, détermine la plage d'opération et la résolution minimale de chaque opérande à partir du calcul de la moyenne, μ_i et l'écart type, σ_i des valeurs que prend cet opérande durant une simulation de l'algorithme. Quoique plus lente que l'approche par DFG, cette approche est relativement rapide. Cependant cette approche base ses solutions sur un seul opérande à la fois, et non pas sur la combinaison de tous les opérandes simultanément. Sachant que les erreurs engendrées par les représentations en virgule-fixe de tous les opérandes peuvent s'accumuler ou se compenser, cette méthode propose des solutions qui ne rencontrent pas les critères de précision spécifiés par le concepteur matériel.

Finalement, IWL, FWL, et s_i de chaque opérande O_i peuvent être déterminés sur la base de simulations. En calculant les différences entre les simulations virgule-flottante et virgule-fixe, une procédure de recherche tente de minimiser ces différences en modifiant la résolution des opérandes appropriés afin de trouver une combinaison de tailles de chemins de données qui rencontre les critères de précision spécifiés par le concepteur matériel. En comparaison avec les autres approches, l'approche par simulation est la plus précise et la moins restrictive. Dans le passé, cette méthode était prohibitivement exigeante en terme de complexité de

calcul, et pour la plupart des algorithmes de traitement de signaux, cette approche ne pouvait être considérée. Avec les récentes améliorations de la performance des processeurs en terme de puissance de calcul, cette méthode peut maintenant être appliquée à des algorithmes de traitement de signaux de complexité moyenne.

Il arrive fréquemment que l'optimisation de la taille des chemins de données doit rencontrer plusieurs critères de précision (relatif au domaine d'application) et au coût d'implantation matérielle. Par example, le comité responsable des normes de la société IEEE propose dans [18] que l'optimisation de la transformée inverse de cosinus discrète doit s'effectuer sur la base de 5 mesures de précision: l'erreur maximale d'un pixel, l'erreur moyenne d'un pixel, l'erreur moyenne quadratique d'un pixel, l'erreur moyenne sur tous les pixels, et l'erreur moyenne quadratique sur tous les pixels. Cependant, aucune des méthodes proposées dans la littérature et utilisant l'approche par simulation ne suggère une optimisation des tailles des chemins de données en tenant compte simultanément de plusieurs mesures de précision, critères de précision et du coût d'implantation. C'est pourquoi une nouvelle méthode automatique et une nouvelle métrique sont présentées dans ce chapitre.

1.1 Méthode automatique proposée

Tel qu'illustrée à la Figure 1.1, la méthode proposée est constituée de huit étapes.

1. Génération d'un ensemble de données

   Le concepteur matériel doit tout d'abord générer un ensemble de données représentatif du domaine d'opération dans lequel l'algorithme de traitement de signaux doit évoluer. L'ensemble des données doit stimuler suffisamment tous les opérandes $\{O_i\}$ visés par la méthode ainsi que tous les autres chemins

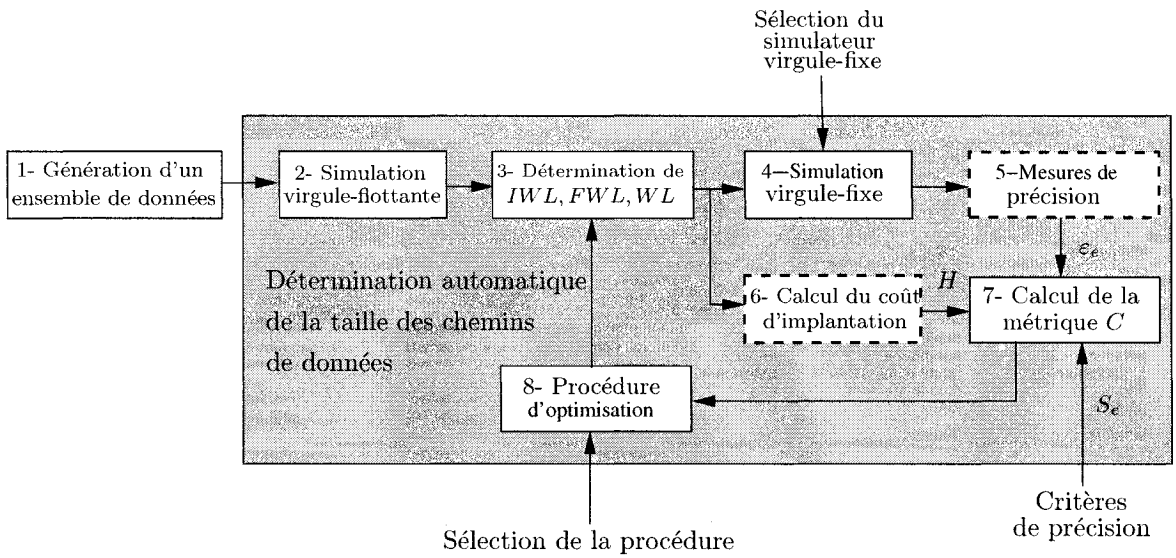


Figure 1.1 Méthode automatique de détermination de la taille des chemins de données

de données d'intérêt.

2. Simulation virgule-flottante

   Une simulation de l'ensemble des données sur l'algorithme de traitement de signaux est effectuée lorsque tous les opérandes $\{O_i\}$, ont une résolution virgule-flottante. Durant cette simulation, plusieurs mesures sur les valeurs que peut prendre chaque opérande sont extraites: la valeur absolue maximale $|O_i|^{\max}$, le signe s_i (Equation 1.1), la moyenne μ_i et l'écart type σ_i. Ces mesures et les résultats de simulation sont conservés comme valeurs de référence pour la suite de la méthode.

3. Détermination de la représentation virgule-fixe

   Pour chaque opérande O_i, la partie entière IWL_i est fixée égale à:

$$IWL_i = \lceil \log_2 (\max (\mu_i + k \times \sigma_i, |O_i|^{\max})) \rceil \tag{1.2}$$

et s_i est fixé égale à 0 si les valeurs que prennent l'opérande O_i sont toujours positives, sinon s_i est fixé égale à 1. La partie fractionnaire FWL_i de chaque opérande est initialisé selon la procédure d'optimisation sélectionnée à l'étape 8 ci-dessous.

4. Simulation virgule-fixe

   Pour l'ensemble des données généré à l'étape 1, une simulation virgule-fixe est effectuée. Les résultats produient par l'algorithme de traitement de signaux sont conservés.

5. Mesures de la précision

   À partir des résultats de simulation conservés aux étapes 2 et 4, plusieurs mesures de précision ε_e relatifs au domaine d'application sont calculés, pour $e = 0, 1, \ldots, E-1$, où E est le nombre de mesures calculées.

6. Calcul du coût d'implantation

   Selon le nombre de bits alloués à chacun des chemins de données, un coût d'implantation H est calculé. Ce coût d'implantation H réflète une mesure de la consommation de puissance, du délais ou de la surface du circuit, que le concepteur matériel désire minimiser.

7. Calcul de la métrique C

   Sur la base des mesures de précision ε_e calculées à l'étape 5, du coût d'implantation H calculé à l'étape 6 et des critères de précision S_e (un seuil minimum de passage est défini pour chaque mesure de précision ε_e), la métrique suivante est calculée:

$$C = \left[\sum_{e=0}^{E-1} \left(\frac{S_e - \varepsilon_e}{S_e}\right) d_e\right] \cdot H + \left[K - \frac{1}{E}\sum_{e=0}^{E-1}\left(1 - \frac{S_e - \varepsilon_e}{S_e}\right)\right] \cdot \frac{1}{H}\prod_{e=0}^{E-1}(1 - d_e) \tag{1.3}$$

où $d_e = 0$ si $S_e \geq \varepsilon_e$, $d_e = 1$ si $S_e < \varepsilon_e$, $K = I \cdot WL^{\max}$, où $WL^{\max}$ est la taille maximale qu'un chemin de données peut prendre dans la simulation virgule fixe. La métrique C grade toutes les combinaisons possibles des tailles des chemins de données vers la solution optimale. Cette métrique qui a été proposée dans [15], sera couverte plus en détails dans la seconde section de ce chapitre.

8. Procédure d'optimisation

Une procédure d'optimisation cherche à trouver une combinaison de taille de chemin de données $\{WL_i\}$ qui maximise la métrique C. Dans le cadre de cette thèse, 4 nouvelles procédures d'optimisation ont été proposées [13][10]: $Min + b$ *bits*, $Max - 1$ *bit*, *Evolutive*, et *Hybride*.

Ces procédures ont été comparées avec 5 autres procédures retrouvées dans la littérature qui ont été implantées et adaptées à la méthode. Cette comparaison s'est tenue sur 2 points de comparaison, soit (i) le nombre de bits nécessaires $\overline{\Delta WL}$[1] des solutions afin de rencontrer les critères de précision spécifiés par le concepteur matériel et (ii) le nombre d'itérations nécessaires $\overline{\Delta N}$ pour trouver ces solutions.

Cette comparaison s'est effectuée sur 12 algorithmes de traitement de signaux, et pour chaque algorithme, chacune des procédures devait tenter de trouver 100 solutions optimales pour 100 différents niveaux de précision. Les Tableaux 1.1 et 1.2 présentent les résultats de cette comparaison. Sachant que ces résultats sont élaborés à partir de la meilleure solution trouvée, les procédures les plus performantes produisent $\overline{\Delta WL}$ et $\overline{\Delta N}$ qui tendent vers 0.

[1]Les variables utilisées dans l'algorithme et citées dans ce résumé sont définies dans l'article qui suit

Tableau 1.1 Tailles des chemins de données obtenues $\left(\overline{\Delta WL}\right)$ par les différentes procédures d'optimisation

Bancs – d'essais	I	Heur. $\overline{\Delta WL}$	Exh. $\overline{\Delta WL}$	Sim. Ann. $\overline{\Delta WL}$	Prep. $\overline{\Delta WL}$	B & B $\overline{\Delta WL}$	Min+ B bit $\overline{\Delta WL}$	Max- 1 bit $\overline{\Delta WL}$	Evol. $\overline{\Delta WL}$	Hyb. $\overline{\Delta WL}$
DSP_1	3	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
DSP_2	3	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
DSP_3	5	0.00	0.61	0.61	0.61	0.72	0.61	0.72	0.72	0.61
DSP_4	7	0.04	0.41	0.19	0.23	0.34	0.19	0.68	0.11	0.04
DSP_5	5	0.00	0.48	0.48	0.56	0.54	0.48	0.67	0.54	0.48
DSP_6	4	0.39	0.00	0.39	0.34	0.00	0.39	0.00	0.00	0.00
DSP_7	6	0.07	0.76	0.42	0.90	0.20	0.63	0.07	0.07	0.04
DSP_8	4	0.03	0.31	0.06	0.18	0.18	0.12	0.06	0.05	0.00
DSP_9	15	0.40	0.00	0.40	0.38	0.18	0.40	0.00	0.00	0.00
DSP_{10}	3	0.22	1.01	0.98	1.30	0.83	1.01	0.52	0.25	0.22
DSP_{11}	36	0.18	1.75	1.53	3.07	1.99	1.65	0.45	0.18	0.21
DSP_{12}	3	0.23	0.11	0.02	0.05	0.03	0.02	0.42	0.04	0.01

À partir de ces deux tableaux tirés de l'article qui suit, il est possible d'observer plusieurs situations:

- Aucune des procédures ne trouve toujours la solution la plus optimisée.
- Pour certains algorithmes (DSP_4, DSP_7, DSP_{10},DSP_{11} et DSP_{12}), aucune procédure ne produit $\overline{\Delta WL} = 0$. Autrement dit, selon la précision spécifiée, ce n'est pas toujours la même procédure qui trouve la solution la plus optimisée.
- Pour certains algorithmes (DSP_1, DSP_3, DSP_4, DSP_5, DSP_7, DSP_8, DSP_{10} et DSP_{11}) aucune procédure ne produit $\overline{\Delta N} = 0$, car ce n'est pas toujours la même procédure qui trouve une solution avec le moins d'itérations.

En considérant le fonctionnement des procédures, il est observé que:

- Certaines procédures tel que $Max - 1$ *bit* et *Evolutive* qui démarrent leur

Tableau 1.2 Nombre d'itérations obtenus $\left(\overline{\Delta N}\right)$ par les différentes procédures d'optimisation

Bancs – d'essais	I	Heur. $\overline{\Delta N}$	Exh. $\overline{\Delta N}$	Sim. Ann. $\overline{\Delta N}$	Prep. $\overline{\Delta N}$	B & B $\overline{\Delta N}$	Min+ B bit $\overline{\Delta N}$	Max- 1 bit $\overline{\Delta N}$	Evol. $\overline{\Delta N}$	Hyb. $\overline{\Delta N}$
DSP_1	3	0.2	3.2	20.2	1.9	2.1	2.2	70.6	1.3	3.2
DSP_2	3	0.0	1.0	20.0	0.0	2.7	0.0	72.5	4.0	1.0
DSP_3	5	1.0	3.2	52.2	1.7	2.0	2.2	56.0	0.5	3.2
DSP_4	7	1.6	2.1	98.2	0.2	52.1	0.2	37.2	3.4	1.6
DSP_5	5	0.7	3.6	52.0	1.7	5.2	2.0	63.5	0.7	3.0
DSP_6	4	0.0	2.3	40.0	0.0	3.6	0.0	97.5	5.0	2.3
DSP_7	6	0.2	7.5	60.7	0.2	2.4	0.6	134.0	8.2	5.1
DSP_8	4	2.3	2.5	98.8	0.2	4.8	0.2	84.0	4.5	1.6
DSP_9	15	2.6	0.0	450.0	0.0	40.25	0.0	105.9	5.5	2.6
DSP_{10}	3	2.2	4.1	18.4	0.1	4.2	0.4	75.6	5.5	3.7
DSP_{11}	36	2.5	15.5	2593	1.2	2964	1.5	172.9	12.4	12.5
DSP_{12}	3	1.7	0.7	18.2	0.0	7.6	0.2	67.3	2.3	1.3

recherche à partir d'une solution qui rencontre déjà les critères de précision, peuvent être piégées dans un optimum local.

- La procédure *Preplanned* est celle qui la plupart du temps requière le moins d'itérations pour trouver une solution. Cependant, les solutions trouvées par cette procédure ne sont pas souvent les plus optimisées en terme du nombre de bits nécessaires.

- La procédure *Hybride* obtient toujours les mêmes ou de meilleures solutions que la procédure *Min + b bits*. Cette observation démontre qu'il est possible d'obtenir une solution qui contient moins de bits que la combinaison MWL. La combinaison MWL est obtenue lorsque la résolution de chaque opérande est égale au nombre de bits minimum pour rencontrer tous les critères de précision S_e lorsque tous les autres opérandes ont une résolution virgule flottante.

- Les procédures *Exhaustive*, *Preplanned*, *Branch & Bound* et *Min + b bits*

ne considèrent pas dans leur domaine de recherche des solutions avec moins de bits que la combinaison MWL. Cependant, il a été démontré qu'il existe des solutions avec moins de bits que la combinaison MWL.

- Les procédures *Exhaustive*, *Branch & Bound* et $Max - 1$ *bit* exigent de façon significative, beaucoup plus d'itérations pour trouver une solution que les autres procédures. Leur utilisation peut devenir prohibivement complexe en terme de calcul lorsqu'un grand nombre de chemin de données est analysé.

- Les procédures *Hybride* et *Heuristique* sont celles qui trouvent la plupart du temps les solutions qui nécessitent le moins de bits avec un nombre résonnable d'itérations. La Procédure *Hybride* proposée apparaît être une bonne procédure alternative pour trouver rapidement une solution qui rencontre les critères de précision spécifiés par le concepteur matériel.

Comme nous le verrons par la suite aux Chapitre II et III, avec de telles procédures, la méthode proposée permet de trouver rapidement des solutions optimisées et ce, pour un ou plusieurs critères de précision. La méthode permet de réduire le temps de conception, les coûts d'implantation, et la consommation de puissance, tout en favorisant les performances désirées de l'algorithme.

La suite de cette section présente l'article intitulé "An automatic Word Length Determination Method" paru dans *WSEAS Transaction on Information Science and Applications*, Issue 5, Volume 1, November 2004, pp. 1440-1448, et qui décrit plus en détail la méthode proposée.

1.2 Article: An Automatic Word Length Determination Method

MARC-ANDRE CANTIN AND YVON SAVARIA
Electrical Engineering Department
École Polytechnique de Montréal
C.P. 6079, succursale Centre-Ville, Montréal (Quebec), H3C 3A7
CANADA
cantin@grm.polymtl.ca http://www.grm.polymtl.ca

Abstract: - Automatic word length determination of hardware data paths may require considering several error models, user specifications and hardware costs. A new automatic method for determining the word lengths in hardware data paths that considers these requirements is proposed and analyzed. The search-based method uses a C/C++ fixedpoint simulation tool to model the impact of finite word lengths on overall accuracy. By computing dissimilarities between fixed-point and floating-point simulation results, a procedure searches for a combination of word lengths that meets accuracy criteria specified by the designer. This method is presented with four novel maximization procedures. These four automatic procedures are compared with other procedures given in the literature and adapted into the method. The comparison helps to select a procedure that finds a combination of word lengths that meets user specifications, in a small number of iterations. The procedures are characterized and compared using a dozen DSP algorithms.

Key-Words: - Floating-point, Fixed-Point, Automatic Determination, Optimization, Word Length, Comparison.

1.2.1 Introduction

Digital Signal Processing (DSP) algorithms are often expressed in 32- or 64-bit fixed- or floating-point data formats since these are available in most commercial off-the-shelf processors. Yet, in spite of very significant improvements in processor performance and power efficiency, the requirements of many real-time applications in terms of pure performance or low power operation command the use of specialized hardware [1]. Since cost, power dissipation, and performance are highly dependent on the number of bits used to represent data, and on the use of floating-point operators, the problem of precise word length determination is important. In some complex designs, half of the design time can be spent determining word lengths [2]. Moreover, many algorithms require a careful selection of word lengths to preserve their stability [3]. Therefore, powerful automatic word-length determination methods are required.

For some specific applications, the resulting word length combination found by these methods, must meet various error specifications [4][5]. Furthermore, depending on the application, the resulting word length combination must optimize one or more performance metric such as hardware cost, latency, throughput, area, or power consumption. Especially, for search-based maximization procedures, it is important to grade all word length combinations to guide the search toward the optimal solution. However, none of the methods found in the literature propose a way to handle several error models, user specifications, and implementation cost models in a common optimization process. Thus a new automatic word length determination method, which is based on a recently proposed metric, is proposed.

This paper is structured as follows. Methods found in the literature and previous works on word length determination are reviewed in Section 1.2.2. A new automatic word length determination method that considers these requirements

is presented and described in Section 1.2.3, together with four novel search procedures. In Section 1.2.4, these four procedures are compared with procedures found in the literature. The methodology and results obtained in this comparative study are presented in the same section. The main conclusions are summarized in Section 1.2.5.

1.2.2 Related Work

In the last 10 years, several techniques have been proposed to translate floating-point formulations into fixed-point formulations, especially for specific DSP applications [3][4][6]. Heuristic techniques and analytical methods dedicated to specific applications have been employed. For general DSP applications, the translation of floating-point formulations into fixed-point formulations consists of evaluating the dynamic range and the minimum accuracy, by determining the Integer Word Length (IWL) and the Fractional Word Length (FWL) of each operand O_i, $i = 0, 1, \ldots, (I-1)$, where I is the number of operands to be translated. The Word Length (WL) of each translated fixed-point operand may then be obtained as follows:

$$WL_i = IWL_i + FWL_i + s_i \tag{1.4}$$

where $s_i = 0$ for unsigned and $s_i = 1$ for two's complement representations of O_i. In Equation (1.4), the IWL and FWL can be either positive or negative. For example, a 4-bit binary data 1010 with $s_i = 0$ and $IWL_i = -2$ should be interpreted as 0.001010, which corresponds to a real value of 0.15625. The same 4-bit binary data with $s_i = 0$ and $FWL_i = -2$ will be interpreted this time as 101000, which corresponds to a real value of 40. To ensure that the WL is greater

than, or equal to 0, the sum of IWL and FWL must be greater than, or equal to 0; $IWL + FWL \geq 0$.

The required Integer Word Length can be estimated using 3 methods. The first method computes the IWL_i as follows:

$$IWL_i = \lceil \log_2 (|O_i|^{\max}) \rceil \tag{1.5}$$

where $\lceil \cdot \rceil$ is the "ceiling" function, and where $|O_i|^{\max}$ is the maximum absolute value of the operand O_i extracted during simulations of the DSP algorithm [7][8]. The second method computes each IWL_i like the first method, except that the maximum and minimum values of the operand O_i are replaced by the values of O_i obtained by propagation of the maximum and minimum input values through a Data Flow Graph (DFG) of the DSP algorithm [9]. The third method, introduced by Sung *et al.* [10], extracts the mean μ_i and the standard deviation σ_i of the operand O_i during a simulation of the DSP algorithm, and then computes the IWL_i as follows:

$$IWL_i = \lceil \log_2 (\max (\mu_i + k \times \sigma_i, |O_i|^{\max})) \rceil \tag{1.6}$$

where k is set equal to 4 in [10]. The first method does not guarantee that overflow will not occur in the data path. On the other hand, the second method is very fast, but it is also known to be a conservative approach that tends to overestimates IWL_i [11], even though it can also underestimate it if partial results cancel out.

Finally, the third method minimizes the probability of overflow in the data path when k is set to a high value.

After the Integer Word Length, it is, the minimum accuracy, or minimum Fractional Word Length (FWL) of each operand O_i that is determined. Like the IWL, the FWL can be estimated by DFG analysis. FRIDGE [2] and CoCentric [12] propagate the $FWLs$ through the DFG, while ensuring that no information is lost. This technique is called interpolation, and may be illustrated by a simple example: for a = b + c, no information is lost if $FWL_a = \max(FWL_b, FWL_c)$.

Alternatively, Yasuura *et al.* [8] and Wadekar [9] perform numerical analysis on the DFG of DSP algorithms.

Determining the FWL using a DFG is fast, but the method overestimates the FWL like it does for the IWL. Furthermore, the DFG analysis requires a fixed-point specification of the input signals, and becomes complex to manage when the algorithm contains data dependencies.

Cmar *et al.* [11] recently proposed a method that combines analytical rules and simulations. The method determines FWL_i by measurement of the mean μ_i and standard deviation σ_i of the dissimilarities between fixed-point and floating-point simulations.

The C language description of the DSP algorithm requires major modifications for the method proposed by Cmar *et al.*, and the bit resolution analysis is performed only on one operand at a time, instead of on combinations of operands.

Finally, the FWL can be determined by simulations. Sung & Kum [13], Han *et al.* [14], Choi and Burleson [15] and Fiore and Lee [16] proposed manual procedures and guidelines to optimize the FWL. These procedures and guidelines that use a search-based methodology are now examined.

A procedure called *Sequential Search* [14] proceeds in three phases. In the

first phase, for each operand O_i, $i = 0, 1, \ldots, (I-1)$, a minimum bit resolution is found such that the user specification is met, when all other operands $O_j, j = 0, 1, \ldots, (I-1), j \neq i$, are in floating-point format. In the second phase of this procedure, the resolution of each operand is set equal to the value found in the first phase. This combination of word lengths is called the Minimum Word Length (MWL) combination. In the third phase, an iterative competition takes place between the operands to gain one bit. A bit is temporarily assigned to each operand, and the configuration that produces the least dissimilarities with a floating point simulation is the one that wins the right to keep the bit. Subsequent competitions take place until the user specifications are met.

The *Heuristic* procedure [13] iteratively increases all word lengths by one bit from the MWL combination until the user specifications of the system are met. At that point, the operand that generates the largest cost savings wins the right to loose a bit as long as the error specifications continue to be met.

The *Exhaustive* procedure [13], as its name suggest, carries out an exhaustive search, starting from the MWL combination. The exhaustive search temporarily assigns B additional bits over the operands. Every possible assignment combination is tried. If the system specifications are not met, then B is increased, that is B =1, 2, 3, ... bits, until one assignment combination meets the system specifications.

A complex procedure for word length determination is proposed in [16]. It uses an approach similar to the classical simulated annealing algorithm [17]. The *Simulated Annealing* procedure only considers solutions that meet the error specifications, but may temporarily explore some solutions that increase the total system hardware cost. The procedure starts with a word length combination such that the overall solution meets the system specifications. Then some WL_is are increased in order

to allow reducing other WL_js, $j \neq i$, in an attempt to minimize the total system hardware cost. The user determines how many times this step is repeated.

The *Preplanned* procedure [14] proceeds in four execution phases. The first phase computes the system performance for each operand O_i, $i = 0, 1, \ldots, (I-1)$, when WL_i is set equal to every possible bit resolution, $WL_i = 0, 1, \ldots, (WL^{\max} - 1)$, where $(WL^{\max} - 1)$ is the maximum accuracy supported by the fixed point simulation tool. During this first phase, all other operands $O_j, j \neq i$, are in floating-point format, and a sensitivity parameter, ξ_i, is computed for each operand:

$$\xi_i = f(WL_i + 1) - f(WL_i) \tag{1.7}$$

where $f(.)$ is an objective function. The second phase constructs a global priority list in decreasing order of sensitivity. The third phase computes and sets all the O_i to the MWL bit resolution. The last phase is an iterative process. The width of the operand O_i that has the largest sensitivity is increased by one bit. As long as the error specifications of the system are not met, this iterative process is repeated.

The *Branch and Bound* procedure [15] proceeds in 3 execution phases. The first phase finds a minimum uniform word length combination, called the upper bound solution. The second phase computes the MWL combination. The third phase tries all word length combinations exhaustively in the search space between the upper bound and the MWL combination and keeps the best solution found. Limiting the search space to this range tends to drastically prune the search space as compared to a brute force exhaustive search.

1.2.3 An Automatic Optimization Method

Modeling dissimilarities between floating-point and fixed-point simulation results of a DSP algorithm may require computing several error models. For example, the CAS standards Committee of the IEEE Circuits and Systems Society proposes in [5] five error models for the Inverse Discrete Cosine Transform (IDCT) algorithm: the pixel peak error, the pixel mean square error, the pixel mean error, the overall mean square error and the overall mean error. However, for all the simulation-based procedures cited in Section 2, there is no method in the literature that allows handling several error models, user specifications and hardware cost models, in a single unified word length optimization process. Thus, an automatic word length determination method is proposed, and the metric on which it is based is presented in this section. Four novel procedures adapted to the method are also introduced.

1.2.3.1 Description of the Automatic Method

The proposed method is built upon a fixed-point simulation tool. Two fixed-point simulation tools were successfully used: SystemC [18] and a fixed-point utility developed by W. Sung *et al.* [10]. These fixed-point simulation tools convert a floating-point DSP program written in C or C++ into a fixed-point equivalent description. Fixed-point arithmetic operations, instead of floating-point arithmetic, are conducted automatically due to the operator overloading capability of the C++ language. Except for declaring which operands belong to a fixed-point class type, renaming the main function and adding a fixed-point header file, no other part of the original C program needs to be changed to use the proposed method.

The proposed method is composed of eigth steps, as shown in Figure 1.2.

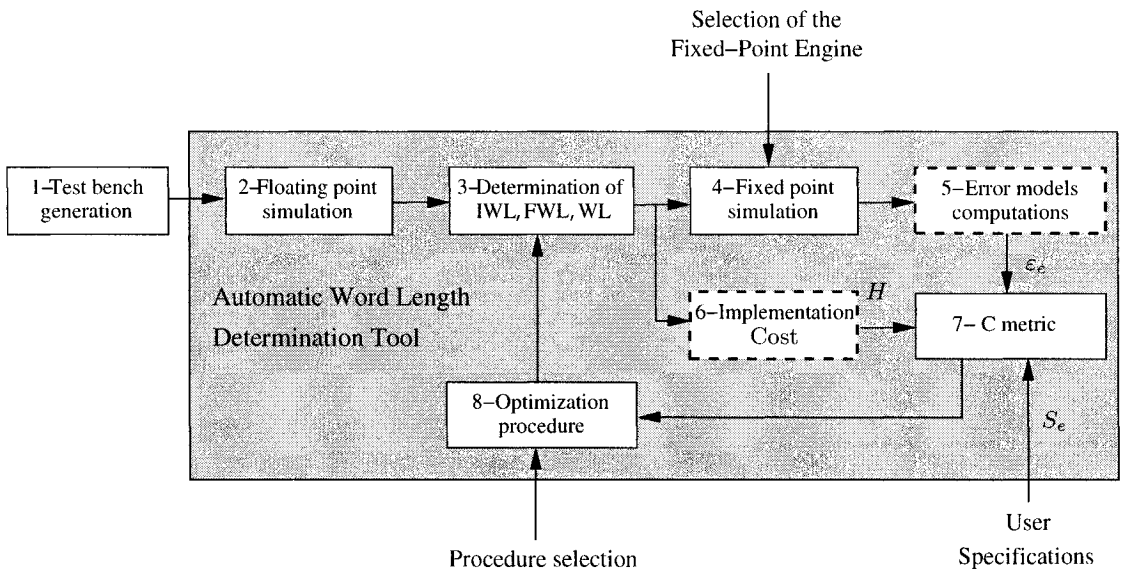


Figure 1.2 Proposed method for the automatic determination of word lengths

1- Test bench generation

The user generates test benches representative of the application. They must stimulate all operands, $\{O_i\}$, in the DSP algorithm, and all data paths of interest. The method considers I user-specified operands, and outputs a word length for each one. Each test bench is a file containing inputs to stimulate the DSP algorithm.

2- Floating-point simulation

A floating-point simulation of the DSP algorithm is performed for each test bench. Several parameters for each operand O_i are extracted and computed from the simulation: the maximum absolute value $|O_i|^{\max}$, the sign flag, s_i (see Equation (1.4)), the mean μ_i, and the variance σ_i. These parameters and output results produced by the DSP algorithm in floating-point resolution are recorded for future reference.

3- Determination of the fixed-point formats.

For each operand O_i, the integer word length IWL_i is set equal to Equation (1.6), and s_i to 0 if the operand O_i is always positive, and to 1 otherwise. The Fractional Word Length FWL_i is initialized according to the selected maximization procedure to be run in Step 8 below.

4- Fixed-point simulation

A fixed-point simulation is performed for each test bench. Output results produced by the DSP algorithm in fixed-point resolution are recorded.

5- Error computation

Floating-point and fixed-point simulation results stored in Steps 2 and 4 respectively are used to compute the dissimilarity between the floating- and fixed-point models according to various error ε_e, for $e = 0,\ 1,\ \ldots,\ E-1$ where E is the number of error models. Error models, ε_e, such as the maximum absolute error and mean square error between each corresponding floating- and fixed-point output, are provided to quantify the dissimilarities between the floating- and fixed-point models. However the user is also allowed to provide his error computations ε_e by using predetermined C functions to get the floating- and fixed-point output results and return the resulting ε_e values.

6- Implementation cost

The implementation cost, denoted by H, is some measure of the hardware cost, power dissipation, or processing time for the word length combination $\{WL_i\}$, and is computed in the range $[1,\ \infty[$. By default, the implementation cost H is defined as follows:

$$H = \sum_{i=0}^{I-1} WL_i \tag{1.8}$$

where WL_i is the total length in bits of the i^{th} operand. Since Equation (1.8) is a simplification of reality and with no loss of generality, the user is allowed to provide his own model to compute the implementation cost H based on the word length WL_i of each operand O_i.

7- Computation of the C metric

Based on the error computations ε_e produced in Step 5, the implementation cost H produced in Step 6, and the user specifications S_e (a pass/fail threshold for each error computation ε_e), the following C metric recently proposed in [19], is computed.

$$C = \left[\sum_{e=0}^{E-1} \left(\frac{S_e - \varepsilon_e}{S_e}\right) d_e\right] \cdot H + \left[(I \cdot WL^{\max}) - \frac{1}{E} \sum_{e=0}^{E-1} \left(1 - \frac{S_e - \varepsilon_e}{S_e}\right)\right] \cdot \frac{1}{H} \prod_{e=0}^{E-1} (1 - d_e) \tag{1.9}$$

where $d_e = 0$ if $S_e \geq \varepsilon_e$, and $d_e = 1$ if $S_e < \varepsilon_e$. Note that Equation (1.9) is composed of two terms that are mutually exclusive as a function of whether the specifications are met (left side) or not (right side). The C metric grades all word length combinations and can be used to guide a search procedure towards the optimal solution with the term $\sum_{e=0}^{E-1} \left(\left(S_e - \varepsilon_e\right)/S_e\right)$ found in both parts of the metric. User specifications are normalized to provide an equal weight in the metric for each part. Thus, for two word length combinations having the same implementation cost, the one that minimizes dissimilarities between the floating-point and fixed-point simulations yields a higher value of the metric. For two word

length combinations producing the same output accuracy, the one that minimizes the implementation costs yields a higher value of the metric. This behavior comes from the implementation cost term H found in both parts of the metric. The maximum value of the metric is obtained at the optimal solution.

8- Maximization procedure

A maximization procedure tries to find the combination of word lengths $\{WL_i\}$ that maximizes the metric C as defined above. In the following sub-section, 4 new procedures are proposed.

1.2.3.2 Maximization Procedures

In this work, four maximization procedures are examined and compared.

a) A procedure called *Min+B bit* is an extension to the procedure called *Sequential Search* [13]. It proceeds in three phases. The first two phases remain the same as those of the *Sequential search* procedure. In the third phase of the *Min + B bit* procedure, an iterative competition takes place between the operands to gain B bits. For some DSP algorithms, it was found that increasing the bit resolution of a single operand at a time by 1 bit could fail to converge to an acceptable solution. This happens, for instance, when the overall accuracy only increases if two or more operands are simultaneously widened. Therefore, when adding 1 bit fails to provide significant accuracy improvements, $2, 3, \ldots, B$ bits are temporarily distributed on the I operands until the error decreases. At all steps, the procedure explores all L possible ways of assigning B bits to I operands, where L are the values defined by the Pascal triangle (see Equation (1.10)). The value of B is kept as small as possible, and is increased only as needed to reduce the error. Once the error is reduced, B is reset to 1, and the iterative competition is repeated until the user

specifications are met.

$$L = \frac{(I+B-1)!}{(I-1)!\,(B)!} \tag{1.10}$$

b) A procedure called by $Max-1$ *bit* starts with the maximum bit resolution, $WL^{\max}-1$, allowed by the fixed-point simulation tool for all operands. Then the operands compete to loose their bits as follows. One bit is temporarily removed from each operand O_i, while all other operands O_j, $j \neq i$, remain unchanged. The operand O_i for which the C metric is maximized wins the competition and looses its bit. The process is repeated for another bit, and so on, as long as the error specifications are met.

With the $Max-1$ *bit* procedure, the final solution is found when removing 1 bit of any single operand at a time fails to meet the error specification. However for some DSP algorithm, if two or more operands are simultaneously shortened, a better solution may be found. This could happen when the quantization error of more than 2 operands compensate each other. Therefore when removing 1 bit fails to find a better solution, 2, 3, ..., B bits can be removed simultaneously over the I operands. It is not clear what limit should be used for B, and this issue was left for future research, thus the $Max-B$ *bits* procedure was not explored further.

c) An *Evolutive* procedure starts with all operands having floating-point precision. The word length of a first operand, say O_i, is set equal to $WL^{\max}-1$ and gradually decreased as long as the system meets the error specifications. The value of WL_i is then increased by one additional bit, and a second operand is analyzed in the same way. This is repeated until all WL_j are fixed. The user is allowed to determine the order in which the operands are processed. Because the *Evolutive* procedure tends to minimize the first operands processed, it is recommended to process first the

operand having the highest hardware cost and end with the operand having the lowest hardware cost. Thus, it is up to the user to define the order in which the operands are analyzed by determining their declaration order in the C language DSP program.

d) A *Hybrid* procedure combines the *Min+B bit* followed by a minimization of the word lengths as performed in the $Max-1$ *bit* procedure. For some DSP algorithms, it was found that a higher value of the metric C can be obtained by using both procedures together than if either one is used independently.

1.2.4 Comparison

The set of procedures implemented for comparison includes the *Heuristic*, *Exhaustive*, *Simulated Annealing*, *Preplanned*, *Branch and Bound*, *Min+B bit*, $Max-1$ *bit*, *Evolutive* and *Hybrid* procedures. The *Sequential Search* [14] procedure was not considered in the comparison because it sometimes fails to converge to a solution, and corresponds to the *Min+B bit* procedure with $B = 1$.

In the *Simulated Annealing* procedure, the O_is must be initialized to values that already meet the system specifications. The *Min+B bit* procedure was used to find this initial starting point. In the *Simulated Annealing* procedure, the best solution found after $10 \times I$ annealing phases, where that I is the number of operands, is kept as the final solution. In the *Preplanned* procedure, only the required system performances, and then the sensitivity performances are computed instead of all system performances. The upper bound solution of the *Branch and Bound* procedure is computed using a dichotomic search algorithm to reduce the number of iterations. For the procedures that require the MWL combination, this combination is found by using a dichotomic search for each operand. For the heuristic procedure, the

$Max - 1$ *bit* procedure was selected to reduce the word length of operands whose have the highest cost per bit. All the procedures were adjusted in order to use the C metric, making them fully automatic procedures able to handle simultaneously several error models, user specifications and hardware costs.

We applied the nine-optimization procedures to the determination of word lengths for 12 DSP algorithms [20]. The algorithms include the four elementary operations ($+$, $-$, $\times$, $\div$), the fifth order elliptic FIR filter [21], another FIR filter, an IIR filter, an adaptive filter, the CORDIC algorithm [22], the IDCT algorithm [23], a frequency estimation algorithm [24], and a neural network algorithm [25] and are denoted by DSP_1 through DSP_{12}, respectively. For the filters, the word lengths of both coefficients and data-paths were analyzed. The hardware architecture and operands for the fifth order elliptic filter were taken from [12], the IDCT from [26], the frequency estimation algorithm from [27] and the neural network algorithm from [28].

Relevant error models were selected for each DSP algorithm. For the filters, the fast Fourier transform was selected to compute the accuracy of the output frequency responses. The errors models for the IDCT were taken from the IEEE standards specifications [5]. The Rand measurement [29] was used as a quality metric for clustering produced by the neural network. For the other DSP algorithms, the maximum and mean square errors were used.

For the IDCT, the characteristics of the inputs presented were specified in [5]. For the remaining 11 DSP algorithms, the test bench consisted of applying 10000 pseudo-random inputs.

Word lengths were found for the 12 DSP algorithms by the 9 maximization procedures with using K=100 different user specifications. For each procedure, two

results are given in Table 1.3. The first one, $\overline{\Delta WL}$, is a sum of word length differences averaged over K.

$$\overline{\Delta WL} = \frac{1}{K} \sum_{k=1}^{100} \Delta WL_k \tag{1.11}$$

where

$$\Delta WL_k = \frac{1}{I} \sum_{i=0}^{I-1} WL_{ki} - WL_{ki}^{\text{opt}} \tag{1.12}$$

and where WL_i^{opt} is the word length of the operand O_i produced by the procedure that obtained the best word length combination. $\overline{\Delta WL}$ is normalized by the number of operands I. The second result reported in Table 1.4, $\overline{\Delta N}$, is the difference between the number of iterations, N, required to obtain a solution and N^{opt}, the number of iterations required by the procedure that obtained a solution with the lowest number of iterations (generally, the procedure producing N^{opt} does not correspond to the procedure that produce WL_i^{opt}). Note that $\overline{\Delta N}$ is normalized by the number of operands I_n that is,

$$\overline{\Delta N} = \frac{1}{K} \sum_{k=1}^{100} \Delta N_k \tag{1.13}$$

where

$$\Delta N_k = \frac{1}{I_n} \left(N_k - N_k^{\text{opt}} \right) \tag{1.14}$$

If a procedure produces $\overline{\Delta WL} = 0$ and $\overline{\Delta N} = 0$, then it compares favorably to all

other procedures. For DSP_1 and DSP_2, all the procedures found the optimal word lengths. No procedure was able to find the optimal word length for all analyzed DSP algorithms. For some algorithms (DSP_4, DSP_7, DSP_{10}, DSP_{11} and DSP_{12}) no procedure produced $\overline{\Delta WL} = 0$. This corresponds to the situation where different procedures find the optimal solution for different user specifications. By analogy, this situation occurs for DSP_1, DSP_3, DSP_4, DSP_5, DSP_7, DSP_8, DSP_{10} and DSP_{11} for $\overline{\Delta N}$. Note that a difference that may appear small in $\overline{\Delta WL_i}$, for example 1.65 for the *Min+B bit* procedure applied to DSP_{11}, may correspond to a maximum difference as large as 38 bits in total operand widths, when comparing a solution to the optimal solution for some system specification. Since the number of iterations required to find a solution dominates the processing time, a small difference in $\overline{\Delta N}$ is not very significant when a small number of operands are processed.

For instance, in some applications, up to 1000 operands are processed [2]. A difference of $\overline{\Delta N} = 10$ for example would imply $10 \times 1000 \times 0.75\text{s} = 2.08$ additional hours of processing time if 750ms were required to performed one fixed-point simulation of relevant test cases. $\overline{\Delta N}$ significantly larger than 10 have been observed. This may translate in very long additional processing time.

The *Hybrid* procedure always reaches a solution equivalent or better than the *Min+B bit* procedure, resulting in lower hardware cost. By analyzing the details of the simulations, we found two explanations: 1) An optimal solution can be reached with less hardware cost than the MWL combination. This counterintuitive result was observed several times when the quantization errors contributed by two operands or more compensate each other. 2) Finding a solution with the *Min+B bit* procedure does not ensure that all operands have their minimum word length. Therefore, for both situations, the *Hybrid* procedure takes advantage of using the $Max - 1$ *bit* procedure. However since the *Hybrid* procedure adds steps to the *Min+B bit* procedure, it obviously requires additional iterations.

Procedures such as the *Max − 1 bit* and the *Evolutive* start from a solution that already meets the system error specifications, and then try to find a better solution. They can be trapped in a local optimum instead of finding the global optimum, since the search domain is not monotone. The *Heuristic* procedure produced optimized solutions with a relatively small number of iterations. The *Simulated Annealing* procedure always produced the same solutions as the *Min+B bit* procedure, and therefore it does not appear to bring any advantage, at least for the DSP algorithms considered here.

Most of the time, the *Preplanned* procedure required the smallest number of iterations to find a solution. However, the solutions it produces are not always the best in terms of hardware cost. Moreover, the *Preplanned* procedure, as well as the *Min+B bit* and the *Branch and Bound* procedures, do not consider solutions that require less hardware than the MWL combination that were found feasible in some cases. The *Branch and Bound*, *Exhaustive* and *Max − 1 bit* procedures need a large number of iterations to find a solution. They may become prohibitive when a problem with a large number of operands is analyzed.

From these observations, it is found that the *Heuristic* and the new *Hybrid* procedures are very good solutions, albeit not always optimal. The *Hybrid* procedure often produces solutions with less hardware cost, at the expense of additional iterations. This hybrid procedure, proposed by the authors, appears to be a good alternative for rapidly finding a combination of word lengths that meets user specifications.

With these procedures, the method finds rapid and accurate solutions for several user specifications. It reduces the design time, implementation costs, and power dissipation, and it also allows performance increases. Furthermore, the method can be used by procedures already proposed in the literature, and it makes them

Tableau 1.3 Results of the comparative study $\left(\overline{\Delta WL}\right)$

Bench – marks	I_n	Heur. $\overline{\Delta WL}$	Exh. $\overline{\Delta WL}$	Sim. Ann. $\overline{\Delta WL}$	Prep. $\overline{\Delta WL}$	B & B $\overline{\Delta WL}$	Min+ B bit $\overline{\Delta WL}$	Max- 1 bit $\overline{\Delta WL}$	Evol. $\overline{\Delta WL}$	Hyb. $\overline{\Delta WL}$
DSP_1	3	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
DSP_2	3	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
DSP_3	5	0.00	0.61	0.61	0.61	0.72	0.61	0.72	0.72	0.61
DSP_4	7	0.04	0.41	0.19	0.23	0.34	0.19	0.68	0.11	0.04
DSP_5	5	0.00	0.48	0.48	0.56	0.54	0.48	0.67	0.54	0.48
DSP_6	4	0.39	0.00	0.39	0.34	0.00	0.39	0.00	0.00	0.00
DSP_7	6	0.07	0.76	0.42	0.90	0.20	0.63	0.07	0.07	0.04
DSP_8	4	0.03	0.31	0.06	0.18	0.18	0.12	0.06	0.05	0.00
DSP_9	15	0.40	0.00	0.40	0.38	0.18	0.40	0.00	0.00	0.00
DSP_{10}	3	0.22	1.01	0.98	1.30	0.83	1.01	0.52	0.25	0.22
DSP_{11}	36	0.18	1.75	1.53	3.07	1.99	1.65	0.45	0.18	0.21
DSP_{12}	3	0.23	0.11	0.02	0.05	0.03	0.02	0.42	0.04	0.01

Tableau 1.4 Results of the comparative study $\left(\overline{\Delta N}\right)$

Bench – marks	I_n	Heur. $\overline{\Delta N}$	Exh. $\overline{\Delta N}$	Sim. Ann. $\overline{\Delta N}$	Prep. $\overline{\Delta N}$	B & B $\overline{\Delta N}$	Min+ B bit $\overline{\Delta N}$	Max- 1 bit $\overline{\Delta N}$	Evol. $\overline{\Delta N}$	Hyb. $\overline{\Delta N}$
DSP_1	3	0.2	3.2	20.2	1.9	2.1	2.2	70.6	1.3	3.2
DSP_2	3	0.0	1.0	20.0	0.0	2.7	0.0	72.5	4.0	1.0
DSP_3	5	1.0	3.2	52.2	1.7	2.0	2.2	56.0	0.5	3.2
DSP_4	7	1.6	2.1	98.2	0.2	52.1	0.2	37.2	3.4	1.6
DSP_5	5	0.7	3.6	52.0	1.7	5.2	2.0	63.5	0.7	3.0
DSP_6	4	0.0	2.3	40.0	0.0	3.6	0.0	97.5	5.0	2.3
DSP_7	6	0.2	7.5	60.7	0.2	2.4	0.6	134.0	8.2	5.1
DSP_8	4	2.3	2.5	98.8	0.2	4.8	0.2	84.0	4.5	1.6
DSP_9	15	2.6	0.0	450.0	0.0	40.25	0.0	105.9	5.5	2.6
DSP_{10}	3	2.2	4.1	18.4	0.1	4.2	0.4	75.6	5.5	3.7
DSP_{11}	36	2.5	15.5	2593	1.2	2964	1.5	172.9	12.4	12.5
DSP_{12}	3	1.7	0.7	18.2	0.0	7.6	0.2	67.3	2.3	1.3

able to handle several error models, user specifications and implementation costs. The method provides a platform to compare various word length optimization procedures. It also provides a framework for architectural exploration by hardware designers. Finally, as presented in [30], the proposed method can be used to validate implementations of DSP algorithms.

1.2.5 Conclusion

For the purpose of improving performance, reducing power dissipation and reducing design time, an automatic method to determine word lengths in fixed-point implementations of DSP algorithms has been proposed, implemented and tested. The method, which uses a search-based simulation methodology, computes a single metric for each word length combination according to user specified error models, performance specifications and implementation cost models. Several procedures that try maximizing this metric and finding a combination of word lengths in a minimum number of iterations were proposed and implemented. Representative procedures proposed in the literature to optimize word lengths of DSP algorithms were reviewed, implemented and adapted to the framework of our automatic tool. The hardware costs and number of iterations required by these procedures were compared with those obtained using our 4 novel procedures through testing with a dozen DSP algorithms.

The proposed method handles multiple error and hardware cost models combined with user defined performance specifications in a common optimization process. Finally, the method and associated tools provides a platform to compare various optimization procedures, and a framework for architectural exploration by hardware designers.

1.2.6 Acknowledgements

The authors acknowledge the financial support of the Defense Research Establishment in Ottawa, the Natural Sciences and Engineering Research Council of Canada and the Canada Research Chair program. They also thank Pierre Lavoie for his guidance and the constructive feedback he provided on this research.

References:

[1] DM. Oseli, M. Mraz, and N. Zimic. Design considerations of hardware based fuzzy controllers. *WSEAS Transaction on Electronics*, vol. 3:811–816, 2004.

[2] H. Keding, M. Willems, M. Coors, and H. Meyr. FRIDGE: a fixed-point design and simulation environment. *Design, Automation and Test in Europe*, pages 429–435, 1998.

[3] M.-A. Cantin, Y. Blaquière, Y. Savaria, P. Lavoie, and E. Granger. Analysis of quantization effects in a digital hardware implementation of a fuzzy ART neural network algorithm. *IEEE Int. Symposium on Circuits and Systems*, vol. 3:141–144, 2000.

[4] J. Yli-Kaakinen and T. Saramaki. An efficient algorithm for the design of lattice wave digital filters with short coefficient wordlength. *International Symposium on Circuits and Systems*, vol. 3:443–448, 1999.

[5] CAS Standards Committee of the IEEE Circuits and Systems Society. IEEE Standard Specifications for the Implementations of 8x8 Inverse Discrete Cosine Transform, 1991.

[6] I.-T. Lim and J. Bahn. Optimal wordlength determination of AC-3 decoding hardware based on fixed-point analysis and simulations of AC-3 algorithm. *IEEE Workshop on Signal Processing*, pages 301–310, 1997.

[7] T. Aamodt. Floating-Point to Fixed-Point Compilation and Embedded Architectural Support. Master's thesis, University of Toronto, 2001.

[8] H. Yamashita, H. Yasuura, F. N. Eko, and C. Yun. Variable Size Analysis and Validation of Computation Quality. *Proceedings of Workshop on High-Level Design Validation and Test*, pages 95–100, 2000.

[9] S.A. Wadekar. Accuracy Sensitive Word-length Selection for Algorithm Optimization. *Proceedings of the International Conference on Computer Design: VLSI in Computers and Processors*, pages 54–61, 1998.

[10] S. Kim, Ki-Il Kum, and W. Sung. Fixed-Point Optimization Utility for C and C++ Based Digital Signal Processing Programs. *IEEE Transaction on Circuits and Systems II*, vol. 45(11):1455–1464, 1998.

[11] R. Cmar, L. Rijnders, P. Schaumont, S. Vernalde, and I. Bolsens. A methodology and design environment for DSP ASIC fixed point refinement design. *Automation and Test in Europe Conference and Exhibition*, pages 271–276, 1999.

[12] Synopsys Inc. *Press Release: Synopsys Accelerates System-Level C-Based DSP Design With CoCentric Fixed-Point Designer Tool*, 2000.

[13] W. Sung and K. Kum. Simulation-based word-length optimization method for fixed-point digital signal processing systems. *IEEE Transaction on Signal Processing*, vol. 43(12):3087–3090, 1995.

[14] K. Han, I. Eo, K. Kim, and H. Cho. Numerical word-length optimization for CDMA demodulator. *IEEE International Symposium on Circuits and Systems*, vol. 4:290–293, 2001.

[15] H. Choi and W. P. Burleson. Search-based wordlength optimization for VLSI/DSP synthesis. *VLSI Signal Processing VII*, pages 198–207, 1994.

[16] P.D. Fiore and Li Lee. Closed-form and real-time wordlength adaptation. *Proceedings of the IEEE International Conference on Acoustics, Speech and Signal Processing*, vol. 4:1897–1900, 1999.

[17] S. Kirkpatrick, C.D. Gelatt, and M.P. Vecchi. Optimization by simulated annealing. *Science*, 220:671–680, 1983.

[18] M. Speitel and B. Niemann. SystemC design language for development of ASICs and systems. *Elektronik*, vol. 50(13):78–83, 2001.

[19] M.-A. Cantin, Y. Savaria, D. Prodanos, and P. Lavoie. A Metric for Automatic Word Length Determination of Hardware Datapaths. 5^{th} *Revision Submitted in March 2005 to IEEE Transaction on Computer-Aided Design*, 2003.

[20] M.-A. Cantin, Y. Savaria, and P. Lavoie. A Comparison of Automatic Word Length Optimization Procedures. *IEEE International Symposium on Circuits and Systems*, vol. 2:612–615, 2002.

[21] L. Claesen, F. Catthoor, D. Lanneer, G. Goossens, S. Note, J. Van Meerbergen, and H. De Man. Automatic Synthesis of Signal Processing Benchmark using the CATHEDRAL Silicon Compilers. *Proceedings of IEEE Custom Integrated Circuits Conference*, pages 14.7.1–14.7.4, 1988.

[22] J.E. Volder. The CORDIC Trigonometric Computing Technique. *IRE Transaction on Electronic Computers*, vol.EC-8(3):330–334, 1959.

[23] T. Miyazaki, T. Nishitani, M. Edahiro, and I. Ono. DCT/IDCT processor for HDTV developed with DSP silicon compiler. *Journal of VLSI Signal Processing*, vol. 5:39–47, 1993.

[24] S.N. Crozier. Performance and Complexity of Discrete-Time Frequency Estimation Algorithms. *17th Biennial Symposium on Communications, Queen's University, Ontario, Canada*, 1994.

[25] G.A. Carpenter and S. Grossberg. A massively parallel architecture for a self-organizing neural pattern recognition machine. *Computer Vision, Graphics and Image Processing*, vol. 37:54–115, 1987.

[26] S. Kim and W. Sung. Fixed-Point Error Analysis and Word Length Optimization of 8x8 IDCT Architectures. *IEEE Transaction on Circuits and Systems for Video Technology*, vol. 8(8):935–940, 1998.

[27] L.-P. Lafrance, M.-A. Cantin, Y. Savaria, S.H. Sung, and P. Lavoie. Architecture and performance characterization of hardware and software implementations of the crozier frequency estimation algorithm. *IEEE International Symposium Circuits and Systems*, vol. 4:823–826, 2002.

[28] M.-A. Cantin, Y. Blaquière, Y. Savaria, E. Granger, and P. Lavoie. Implementation of the fuzzy ART neural network for fast clustering of radar pulses. *IEEE International Symposium on Circuits and Systems*, vol. 2:458–461, 1998.

[29] W.M. Rand. Objective Criteria for the Evaluation of Clustering Methods. *Journal of the American Statistical Association*, vol. 66(336):846–850, 1971.

[30] S. Catudal, M.-A. Cantin, and Y. Savaria. Performance Driven Validation Applied to Video Processing. *WSEAS Transaction on Electronics*, vol. 1(3):568–575, 2004.

1.3 Métrique pour la determination automatique de la taille des chemins de données

Tel que démontré dans l'article précédent [10], la méthode repose sur une métrique capable de considérer simultanément, plusieurs mesures de précision, objectifs de précision et les coûts de l'implantation. La métrique grade chacune des combinaisons possibles des tailles des chemins de données, et guide une procédure de recherche vers la solution optimale.

Voici les quatre critères qui ont mené à la formulation de la métrique:

1. La métrique doit considérer plusieurs mesures de précision. $\varepsilon_0, \varepsilon_1, \ldots, \varepsilon_{E-1}$, où E est le nombre total de mesures qui modélisent la différence entre la représentation virgule flottante et virgule-fixe de l'algorithme de traitement de signaux. Pour chaque mesure, la métrique doit considérer un objectif préalablement spécifié par le concepteur matériel.

2. Pour chaque combinaison possible des tailles de chemins de données, la métrique doit être en mesure de considérer n'importe qu'elle expression qui représente les coûts d'implantation matériel.

3. Toutes les combinaisons possibles des tailles de chemins de données doivent être gradées afin de guider la procédure de recherche vers la solution optimale. Pour deux combinaisons différentes ayant le même coût d'implantation, celle qui minimise le plus la différence entre les simulations à représentation virgule flottante et virgule fixe, se doit d'obtenir la valeur la plus élevée de la métrique. Pour deux combinaisons différentes ayant la même précision dans les résultats de simulation, celle qui minimise les coûts d'implantation se doit d'obtenir la valeur la plus élevée de la métrique. La valeur maximale de la métrique doit être obtenue à la solution optimale.

4. La métrique doit générer une valeur négative si au moins un objectif de précision spécifié par le concepteur matériel n'est pas rencontré. Autrement la métrique doit générer une valeur positive.

Sur la base de ces critères, une métrique a été proposée dans [15], et elle se définit comme suit:

$$C = \left[\sum_{e=0}^{E-1}\left(\frac{S_e - \varepsilon_e}{S_e}\right) d_e\right] \cdot H + \left[(I \cdot WL^{\max}) - \frac{1}{E}\sum_{e=0}^{E-1}\left(1 - \frac{S_e - \varepsilon_e}{S_e}\right)\right] \cdot \frac{1}{H}\prod_{e=0}^{E-1}(1 - d_e) \tag{1.15}$$

où $WL^{\max}$ est la taille maximale permise par le simulateur à virgule fixe, I est le nombre de chemin de données à analyser, $d_e = 0$ si $S_e \geq \varepsilon_e$ et où $d_e = 1$ si $S_e < \varepsilon_e$. Le coût de l'implantation matériel, H, est définit comme suit:

$$H = \mathcal{F}(\{WL_i\}) \tag{1.16}$$

où WL_i est la taille en bits du $i^{\text{ième}}$ chemin de données et $\mathcal{F}(\cdot)$ est une certaine mesure de la dissipation de puissance, temps de traitement, surface de silicium, ..., que requière l'implantation matérielle de l'algorithme de traitement de signaux lorsqu'il est implanté avec la combinaison de taille $\{WL_i\}$.

Notons que cette métrique est une heuristique et qu'elle n'a pas d'unité. Elle est composée de l'addition de deux termes qui sont mutuellement exclusifs et qui correspondent à deux régions distinctes: (1) la région où tous les critères de précision sont atteints et (2) la région où au moins un critère de précision n'est pas atteint. Dans chacune des régions, la métrique est continue, en autant que cette propriétée soit aussi observée par les mesures de précision et l'algorithme de traitement de

signal qui sont définis par l'usager.

La suite du chapitre présente la 5^{e} revision de l'article intitulé "A Metric for Automatic Word Length Determination of Hardware Datapaths" soumit à *IEEE Transaction on Computer-Aided Design of Integrated Circuits and Systems.* Cet article décrit plus en profondeur les détails mathématiques et le comportement de la métrique. De plus, cette métrique est appliquée à un filtre elliptique du $5^{\text{ième}}$ ordre en exemple, ce qui permet de bien visualiser le fonctionnement de cette métrique.

L'article démontre entre autre que les erreurs engendrées par les représentations en virgule fixe de tous les opérandes peuvent s'accumuler ou se compenser. Lorsque les erreurs se compensent, une situation contre-intutive est observée: l'ajout de bits supplémentaires à un opérande dans la partie fractionnaire FWL peut diminuer l'erreur mais elle peut aussi l'augmenter. Ceci implique qu'il n'est pas toujours vrai qu'en augmentant le nombre de bits dans un circuit, on augmente sa précision.

Cette situation a été souvent observée lors de l'optimisation de filtres numériques. En général, cette situation peut survenir lorsqu'une donnée prend des chemins de données différents puis est recombinée pour former un résultat de sortie alors que l'un des chemins a inversé le signe de celle-ci.

1.4 Article: A Metric for Automatic Word Length Determination of Hardware Datapaths

M.-A. Cantin, Y. Savaria, D. Prodanos, and P. Lavoie

Abstract

A metric for the automatic determination of word lengths required for implementing Digital Signal Processing (DSP) algorithms is proposed. The metric is capable of handling several error models computed between fixed-point and floating-point simulation results to model the impact of finite word lengths on the overall accuracy. It grades all word length combinations and guides a procedure toward the optimal solution. This metric was implemented in an automatic word length determination tool to guide its search of better hardware implementations. It enables creation of a framework for architecture and platform exploration.

1.4.1 Introduction

Digital Signal Processing (DSP) algorithms are often expressed in 32- or 64-bit fixed- or floating-point data formats, since these are available in most commercial off-the-shelf processors. Yet, in spite of very significant improvements in processor performance and power efficiency, the requirements of many real-time applications in terms of pure performance or low power operation commands the use of specialized hardware. Since cost, power dissipation, and performance are highly dependent on the number of bits used to represent data, and on the use of floating-point operators, the problem of precise word length determination is important. In some complex designs, half of the design time can be spent determining word lengths [1]. Moreover, many algorithms require a careful selection of word lengths to preserve

their stability [2]. Therefore, powerful automatic word-length determination methods are required [3]. Previous works on word length determination are reviewed in Section 1.4.2. A metric that simplifies the automatic determination of word lengths is presented and described in Section 1.4.3.

1.4.2 Related Work

In the last 10 years, several techniques have been proposed to translate floating-point formulations into fixed-point formulations, especially for specific DSP applications [2][4][5]. Heuristic techniques and analytical methods dedicated to specific applications have been employed. For general DSP applications, the translation of floating-point formulations into fixed-point formulations consists of evaluating the dynamic range and the minimum accuracy of each operand O_i, by determining its Integer Word Length (IWL_i) and its Fractional Word Length (FWL_i), where $i = 0, 1, \ldots, (I-1)$, and where I is the number of operands to be translated. The resultant Word Length (WL_i) of each translated fixed-point operand O_i may then be obtained as follows:

$$WL_i = IWL_i + FWL_i + s_i \tag{1.17}$$

where $s_i = 0$ if the operand O_i is always positive and $s_i = 1$ otherwise. In Equation (1), the IWL and FWL can either be positive or negative. For example, a 4-bit binary data 1010 with $s_i = 0$ and $IWL_i = -2$ should be interpreted as 0.001010, which corresponds to a real value of 0.15625. The same 4-bit binary data with $s_i = 0$ and $FWL_i = -2$ will be interpreted this time as 101000, which corresponds to a real value of 40. To ensure that the WL is greater than, or equal to 0, the sum of IWL and FWL must be greater than, or equal to 0; $IWL + FWL \geq 0$.

Three known methods can estimate the required IWL. The first method computes

the IWL_i of each operand O_i as follows:

$$IWL_i = \lceil \log_2 (|O_i|^{\max}) \rceil \tag{1.18}$$

where $\lceil \cdot \rceil$ is the "ceiling" function and $|O_i|^{\max}$ is the maximum absolute value of the operand O_i extracted during simulations of the DSP algorithm [6][7]. The second method computes each IWL_i like the first method, except that the maximum and minimum values of the operand O_i are replaced by the values of O_i obtained by propagation of the maximum and minimum input values through a Data Flow Graph (DFG) of the DSP algorithm [8]. The third method, introduced by Sung *et al.* [9], extracts the mean μ_i and the standard deviation σ_i of the operand O_i during a simulation of the DSP algorithm, and then computes the IWL_i as follows:

$$IWL_i = \lceil \log_2 (\max (|\mu_i| + k \times \sigma_i, |O_i|^{\max})) \rceil \tag{1.19}$$

where k is set equal to 4 in [9].

The first method does not guarantee that overflow will not occur in the data path. On the other hand, the second method is very fast, but it is also known to be a conservative approach that tends to overestimates IWL_i [10], even though it can also underestimate it if partial results cancel out. Finally, the third method reduces the probability of overflow in the data path when k is set to a high value.

Once the IWL is evalued, the FWL can be determined by three basic methods found in the literature. The first method evaluates the FWL by Data Flow Graph (DFG) analysis. FRIDGE [1] and CoCentric [11] propagate the FWLs through the DFG, while ensuring that no information is lost. This technique is called interpolation, and may be illustrated by a simple example: for $a = b + c$, no information

is lost if $FWL(a) = \max(FWL(b), FWL(c))$. Alternately, in [7] and [8] the proposed methods are based on numerical analysis of the DFG. As with the IWL, determining the FWL using a DFG is fast, but it is a conservative approach that overestimates the FWL [10]. Futhermore it requires a fixed-point specification of the input signals, and becomes complex to manage when the algorithm contains data dependencies.

The method proposed in [10] combines analytical rules and simulations. It determines FWL_i by measuring the mean μ_i and standard deviation σ_i of the dissimilarities between fixed-point and floating-point simulations. To apply this method, the C language description of the algorithm requires major modifications. Furthermore, the bit resolution analysis is performed only on one operand at a time, instead of on combinations of operands.

Finally, the FWL can be determined by pure simulation. This method computes the dissimilarities between the floating- and fixed-point simulation results, and based on them, a procedure searches for a combination of word lengths that meets some accuracy criteria specified by the user. Compared to the DFG method, fixed-point specifications of the input signals are not required. The simulation based method can analyze DSP algorithms that contain data dependencies, since the procedure modifies the appropriate word lengths until the fixed-point simulations are sufficiently similar to the floating-point simulations. Finally, the simulation based approach does not overestimate word lengths like the DFG approach does [10], since word lengths are determined on test benches representative of the conditions of operation of the system. In the past, for most applications, this method was prohibitively time consuming. However, with recent improvements in the performance of available processors, it can now be used for small and medium complexity DSP algorithms. Using this class of methods, Sung & Kum [12], Han *et al.* [13], Choi and Burleson [14], Kirkpatrick *et al.* [15], Fiore and Lee [16], and the authors [17],

have proposed manual procedures and guidelines to optimize the FWL.

1.4.3 A Proposed Metric

Modeling dissimilarities between floating-point and fixed-point simulation results of a DSP algorithm may require computing several error models. For example, the CAS standards Committee of the IEEE Circuits and Systems Society proposes in [18] five error models for the Inverse Discrete Cosine Transform (IDCT) algorithm: the pixel peak error, the pixel mean square error, the pixel mean error, the overall mean square error and the overall mean error. However, for all the simulation based procedures cited in Section 1.4.2, there is no metric in the literature that can handle several error models, user specifications and hardware cost models, in a single unified word length optimization process.

A key feature to enable a fully automatic word length determination process consists of basing this process on a metric that grades all word length combinations, in search of an optimal solution. Our objective is to obtain a metric capables of handling several error models, user specifications, and hardware cost models.

The desired metric must meet the following four criteria:

1. The metric shall accept error models $\varepsilon_0, \varepsilon_1, \ldots, \varepsilon_{(E-1)}$, where E is the number of error models that measure the dissimilarity between the outputs of the fixed- and floating-point simulations of the DSP algorithm. For each error model ε_e, the metric shall also accept a user specification S_e, a pass/fail threshold for the measured error model.

2. The metric shall accept and consider any suitable expression to compute implementation costs for each operand O_i analyzed.

3. All word length combinations shall be graded to guide the search of an optimal solution. For two word length combinations having the same implementation cost, the one that minimizes the dissimilarity between the floating-point and fixed-point simulations shall yield a higher value of the metric. For two word length combinations producing the same output accuracy, the one that minimizes the implementation costs shall yield a higher value of the metric. The maximum value of the metric shall be obtained at the optimal solution.

4. The metric shall generate a negative value if at least one user specification is not met. Otherwise the metric shall generate a positive value.

Based on these criteria, the following metric may be formulated:

$$C = \left[\sum_{e=0}^{E-1}\left(\frac{S_e - \varepsilon_e}{S_e}\right) d_e\right] \cdot H + \left[(I \cdot WL^{\max}) - \frac{1}{E}\sum_{e=0}^{E-1}\left(1 - \frac{S_e - \varepsilon_e}{S_e}\right)\right] \cdot \frac{1}{H}\prod_{e=0}^{E-1}(1 - d_e) \tag{1.20}$$

where $WL^{\max}$ is the maximum word length allowed by the fixed-point simulation, I is the number of operands, $d_e = 0$ if $S_e \geq \varepsilon_e$, and $d_e = 1$ if $S_e < \varepsilon_e$. The implementation cost, denoted by H, is defined as:

$$H = \mathcal{F}(\{WL_i\}) \tag{1.21}$$

where WL_i is the total word length in bits of the i^{th} operand, and $\mathcal{F}(\cdot)$ is some measure of the operands' hardware cost, power dissipation, or processing time for the word length combination $\{WL_i\}$, as defined by the user, in the range $[1, \infty[$.

Note that Equation (1.20) is composed of two terms that are mutually exclusive as a

function of whether the specifications are met or not. The expression $\sum_{e=0}^{E-1}\left(\frac{S_e-\varepsilon_e}{S_e}\right)$ found in both terms grades all word length combinations and guides search procedures toward the optimal solution in all cases. Furthermore, contributions of all user specifications are normalized by the metric. The first term of Equation (1.20), $\left[\sum_{e=0}^{E-1}\left(\frac{S_e-\varepsilon_e}{S_e}\right)d_e\right]\cdot H$, can only be null or negative. The metric is negative when at least one user specification is not met by the word length combination ($\varepsilon_e > S_e$). When one or more user specifications S_e are not met, $d_e = 1$, and they contribute to the first term. Otherwise, specifications that are met have no effects on the first term as their $d_e = 0$.

The second term of the metric is positive when all user specifications are met. Otherwise, one of the $\{d_e\}$ is equal to 1, in which case the right term is null. When all user specifications S_e are met, the main objective of the metric is to allow reducing implementation cost, since H divides the right term. Furthermore, for two word length combinations that have the same implementation cost, the one that provides the lowest error has the highest C value. This behavior comes from the term $\left[(I\cdot WL^{\max})-\frac{1}{E}\sum_{e=0}^{E-1}\left(1-\frac{S_e-\varepsilon_e}{S_e}\right)\right]$, where $(I\cdot WL^{\max})$ is a constant that ensures $C > 0$, and where the term $\sum_{e=0}^{E-1}\left(1-\frac{S_e-\varepsilon_e}{S_e}\right)$ decreases for word length combinations that reduce the error.

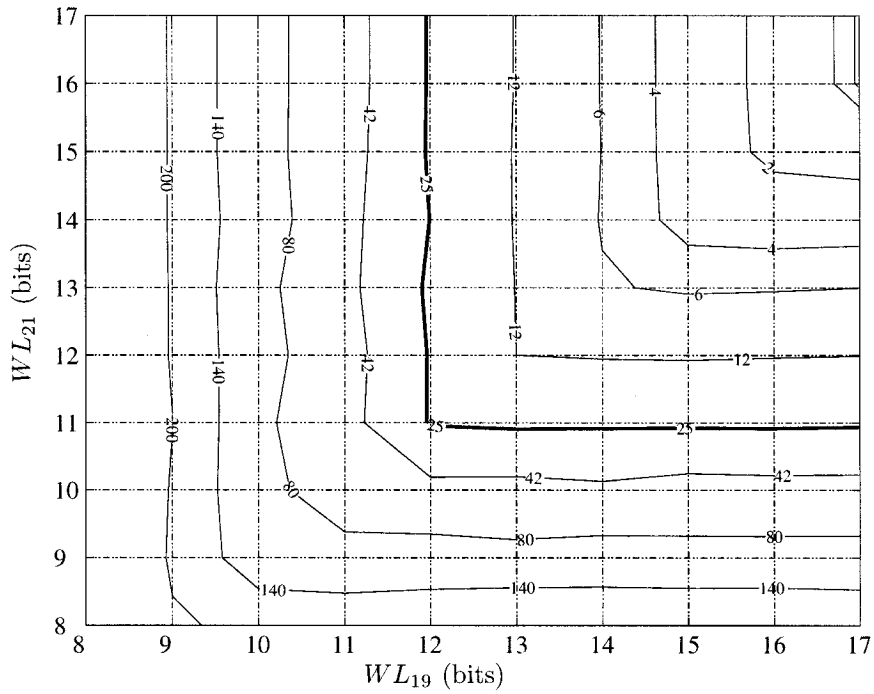
To illustrate how this metric operates, let us examine for example the word lengths of two operands for a fifth order elliptic filter [19], when using two error models and two user specifications. Fig. 1.3. (a) and (b) show the contour curves of the maximum and mean square errors between floating-point and fixed-point simulation results for operands $\{WL_{19}, WL_{21}\}$ [19]. These curves were obtained when random inputs in the range $[-2^{15}, 2^{15}]$ are presented to the input of the filter. Note that only grid points correspond to feasible solutions. Fig 1.3 (a) and (b) were computed by MATLAB that interpolates the feasible solutions to produce contour

plots that are easier to visualize.

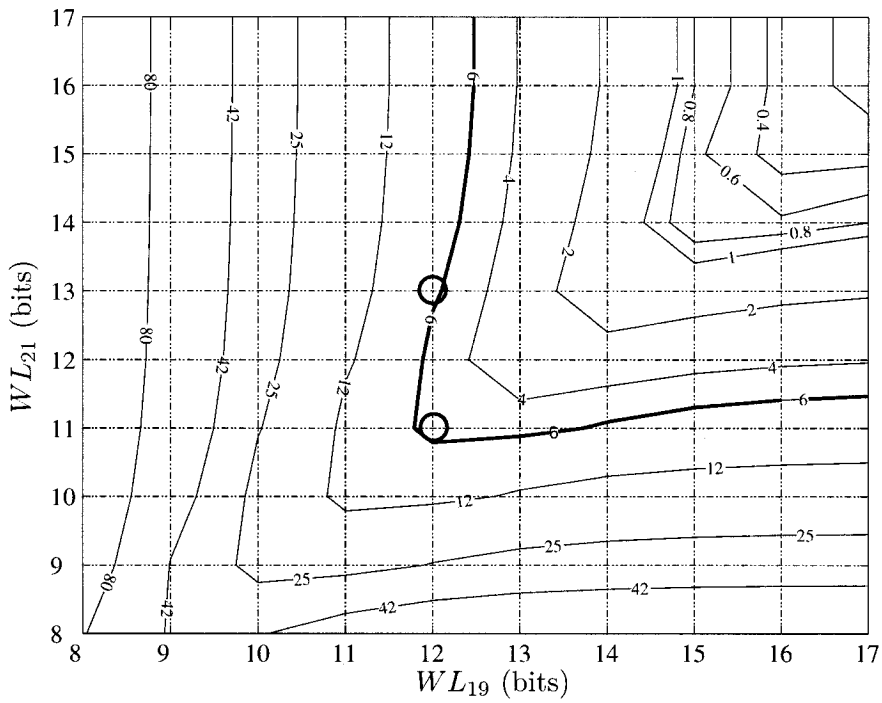
For a maximum absolute error and mean square error of 25 and 6 respectively, the contour curves of the C metric can be computed as illustrated in Fig. 1.4. From this figure, it can be seen that the optimal solution $\{12\text{bits}, 11\text{bits}\}$ is found at the maximal value of the C metric. Two regions can be observed, the one where the C metric value is positive when all user specifications are met, and the one where the C metric value is negative when at least one user specification is not met. For both regions, including the boundary between them, the C metric is continuously differentiable if this propriety is also observed by the error models and DSP algorithm (both user defined), which is a necessary condition for the existence of an optimum.

Fig. 1.4 shows that increasing the word length of an operand O_i can reduce the accuracy of the outputs, which is counterintuitive. For example, if WL_{21} is increased by 2 additional bits from $\{12\text{bits}, 11\text{bits}\}$ to $\{12\text{bits}, 13\text{bits}\}$, then the mean square error is increased from 4.9 to 6.38, and the C metric (see Fig. 1.4) then reduces from 40.1 to -1.6. An analysis of this phenomenon that occurs frequently, in simple and complex algorithms, led us to the following simple example that shows how it can happen. Let us consider a process $a = b - c$, where at equilibrium, the magnitude of b and c should match. If WL_b and WL_c are different, b and c may never match, even though they could if equal values were assigned to b and c. Adding one bit to WL_b or WL_c, without adding it to the other, tends to increase the error over the situation where $WL_b = WL_c$. In general, this type of behavior can happen when some data of opposite sign is processed on two different paths that recombine to produce an output. It was frequently observed when optimizing filters.

The C metric was succesfully implemented in an automatic word length determination tool [3][20]. In order to determine word lengths of a set of operands in the



(a)



(b)

Figure 1.3 (a) Contour curves of the maximum absolute error, for operands O_{19} and O_{21} of the elliptic filter; (b) Contour curves of the mean square error, for operands O_{19} and O_{21} of the elliptic filter;

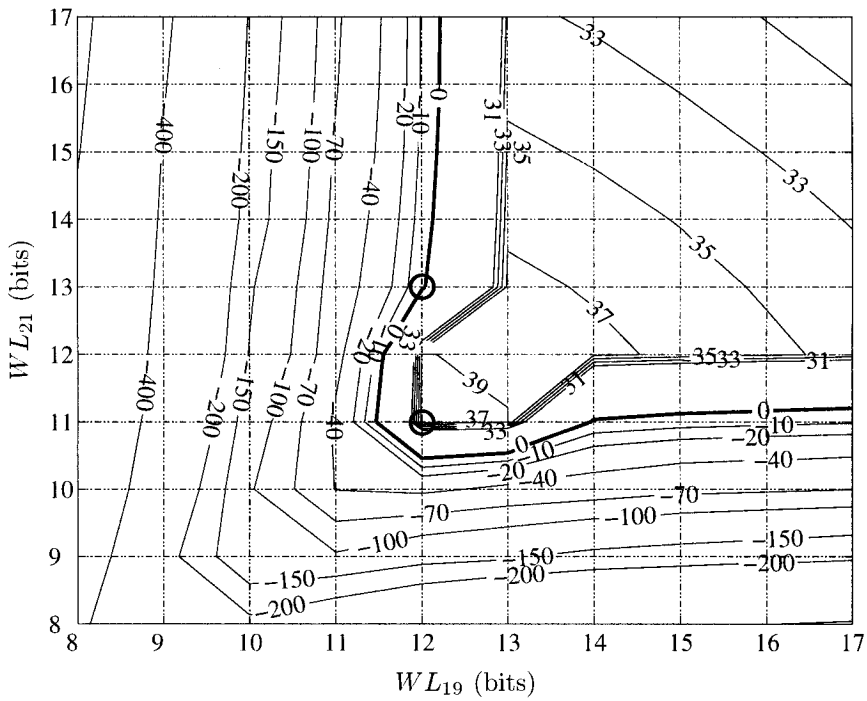


Figure 1.4 Contour curves of the C metric, for operands O_{19} and O_{21} of the elliptic filter

implementation of an algorithm, the user takes its C/C++ software description and only needs to rename the main function, add a fixed-point header file and declare which operands belong to a fixed point class. The description of the algorithm itself is unchanged. This tool was used to compare solutions obtained by 9 search procedures found in the literature for various DSP algorithms [17]. The following discussion focuses on results obtained with this metric for the IDCT and CORDIC algorithms. For the IDCT algorithm, five operands specified in [20] were analyzed. For a maximum pixel peak error of 1, a maximum pixel mean square error of 0.06, a maximum overall mean square error of 0.02, a maximum pixel mean error of 0.015, a maximum overall mean error of 0.0015, and an implementation cost per bit set to 1 for each operand, the automatic method based on this metric found for the test bench specified in [18], that *Dout, Coeff1, Add1, Coeff2*, and *Add2* required word lengths of 19, 16, 20, 16 and 22 bits respectively. The process required 68 iterations with the *Hybrid* search procedure [17], computed in 41.6 minutes when performed on a 1.7GHz Intel Celeron computer running under Redhat Linux 9.0, with 1G bytes of external memory and 128K bytes of cache memory.

For the CORDIC algorithm six operands specified in [21] were analyzed. With a maximum absolute error objective of 0.007, a maximum mean square error objective of 0.0035, $n = 9$ iterations and for a test bench that contains 10^6 pseudo-random angles in the range $[-\pi/2, \pi/2]$, the automatic method found that Z, X, Y, α, $X \cdot 2^{-i}$ and $Y \cdot 2^{-i}$ require word lengths of 11, 14, 13, 9, 13 and 14 bits respectively. The process required 57 iterations with the *Heuristic* search procedure [12] executed in 40.4 minutes on the same computer.

The word length combination solutions found by the method can thus reflect simulated performance when processing real data. The method does not guarantee that the user specifications will be met, or that an overflow in the datapath will not occur with another data set (test bench). Furthermore, making exhaustive simulations

with this time consuming method is often prohibitive. To increase the confidence on the results produced by this method, four strategies are recommended: (i) stimulating the DSP algorithm with pseudo-random inputs, (ii) to avoid overflow in the datapath, verify the mean and standard deviation of each operand and add, if required, additional bits to the IWL, (iii) increase the constraints by reducing the maximum error allowed $\{S_e\}$ by some acceptable margin, and (iv) once a word length combination is found by the method, perform a simulation with more complete test benches to make sure that the user specifications are still met. With these strategies, the method provides rapid and accurate solutions while reducing design time, hardware costs, and power dissipation, as well as allowing performance increases. Finally, the metric can be used for the formal analysis of DSP algorithms as presented in [22] and it enables creation of a framework for architecture and platform exploration by hardware designers.

1.4.4 Conclusion

A metric for the automatic determination of word lengths in fixed-point implementations of DSP algorithms was proposed. The metric can handle several error models, users specifications and implementation costs in a common word length optimization process. It grades all word length combinations and guides a procedure toward the optimal solution. The metric was applied on various DSP algorithms, and guides search procedures found in the literature toward the optimal solution.

References

[1] H. Keding, M. Willems, M. Coors, and H. Meyr, *FRIDGE: a fixed-point design and simulation environment, Design, Automation and Test in Europe*, pp. 429–435, 1998.

[2] M.-A. Cantin, Y. Blaquière, Y. Savaria, P. Lavoie, and E. Granger, *Analysis of quantization effects in a digital hardware implementation of a fuzzy ART neural network algorithm, IEEE International Symposium on Circuits and Systems (ISCAS)*, vol. 3, pp. 141–144, 2000.

[3] M.-A. Cantin, S. Regimbal, S. Catudal, and Y. Savaria. *An Unified Environment to Access Image Quality, Journal of Circuits, Systems and Computers*, vol. 13, no. 6, 2004.

[4] J. Yli-Kaakinen, and T. Saramaki, *An efficient algorithm for the design of lattice wave digital filters with short coefficient wordlength, International Symposium on Circuits and Systems*, vol. 3, pp. 443–448, May 1999.

[5] I.-T. Lim, and J. Bahn, *Optimal wordlength determination of AC-3 decoding hardware based on fixed-point analysis and simulations of AC-3 algorithm, IEEE Workshop on Signal Processing*, pp. 301–310, November 1997.

[6] T. Aamodt, *Floating-Point to Fixed-Point Compilation and Embedded Architectural Support. Masters Thesis, University of Toronto*, 2001.

[7] H. Yamashita, H. Yasuura, F. N. Eko, and C. Yun, *Variable Size Analysis and Validation of Computation Quality, Proc. of Workshop on High-Level Design Validation and Test (HLDVT'00)*, pp. 95–100, November 2000.

[8] S.A. Wadekar, *Accuracy Sensitive Word-Length Selection for Algorithm Optimization, Proceeding of the International Conderence on Computer Design: VLSI in Computers and Processors (ICCD'98)*, pp. 54–61, 1998.

[9] S. Kim, Ki-Il Kum, and W. Sung, *Fixed-Point Optimization Utility for C and C++ Based Digital Signal Processing Programs, IEEE Transaction on Circuits and Systems II*, vol. 45, no. 11, November, 1998.

[10] R. Cmar, L. Rijnders, P. Schaumont, S. Vernalde, and I. Bolsens, *A methodology and design environment for DSP ASIC fixed point refinement, Design, Automation and Test in Europe Conference and Exhibition*, pp. 271-276, 1999.

[11] *Press Release: Synopsys Accelerates System-Level C-Based DSP Design With CoCentric Fixed-Point Designer Tool, Synopsys Inc.*, 2000.

[12] W. Sung, and K. Kum, *Simulation-based word-length optimization method for fixed-point digital signal processing systems, IEEE Transactions on Signal Processing*, vol. 43, no. 12, pp. 3087–3090, 1995.

[13] K. Han, I. Eo, K. Kim, and H. Cho, *Numerical word-length optimization for CDMA demodulator, IEEE International Symposium on Circuits and Systems*, vol. 4, pp. 290–293, 2001.

[14] H. Choi, and W. P. Burleson, *Search-based wordlength optimization for VLSI/DSP synthesis, VLSI Signal Processing*, pp. 198-207, 1994.

[15] S. Kirkpatrick, C.D. Gelatt, and M.P. Vecchi, *Optimization by simulated annealing, Science*, no. 220, pp. 671–680, 1983.

[16] P.D. Fiore, and Li Lee, *Closed-form and real-time wordlength adaptation, Proceedings of the IEEE International Conference on Acoustics, Speech, and Signal Processing*, vol. 4, pp. 1897–900, 1999.

[17] M.-A. Cantin, Y. Savaria and P. Lavoie, *A comparison of automatic word length optimization procedures, IEEE International Symposium on Circuits and Systems (ISCAS)*, vol. 2, pp. 612–615, 2002.

[18] *CAS Standards Committee of the IEEE Circuits and Systems Society, IEEE Standard Specifications for the Implementations of 8x8 Inverse Discrete Cosine Transform*, 1991.

[19] L. Claesen, F. Catthoor, D. Lanneer, G. Goossens, S. Note, J. Van Meerbergen, and H. De Man, *Automatic Synthesis of Signal Processing Benchmark using the CATHEDRAL Silicon Compilers, Proc. of IEEE 1988 Custom Integrated Circuits Conference*, pp. 14.7-14.7.4, 1988

[20] M.-A. Cantin, Y. Savaria, D. Prodanos, and P. Lavoie, *An Automatic Word Length Determination Method, The IEEE Int. Symposium on Circuits and Systems (ISCAS)*, vol. 5, pp. 53-56, 2001.

[21] K. Kota, J. R. Cavallaro, *Numerical Accuracy and Hardware Tradeoffs for CORDIC Arithmetic for Special-Purpose Processors, IEEE Transactions on Computers*, vol. 42, no. 7, pp. 769–779, 1993.

[22] S. Catudal, M.-A. Cantin, and Y. Savaria, *Performance Driven Validation Applied to Video Processing, Word Scientific and Engineering Academy and Society, Transactions on Electronics*, iss. 3, vol. 1, pp. 568-575, 2004.

Tel que mentionné à la fin de cet article, "A Metric for Automatic Word Length Determination of Hardware Datapaths" [15], la métrique a été implantée dans un outil automatique, et a permit de déterminer les tailles des chemins de données de plusieurs algorithmes de traitement de signaux. Cet article met en lumière que l'approche par simulation ne peut pas garantir qu'il n'y aura pas d'erreur de débordement dans les chemins de données et que les critères de précision de l'algorithme seront toujours respectés avec un autre ensemble de données.

Afin de réduire la propabilité que ces problèmes inhérents à l'approche par simulation surviennent, 4 suggestions sont proposées.

i) Stimuler les algorithmes de traitement de signaux avec un ensemble de données générées aléatoirement.

ii) Afin de diminuer les erreurs de débordement, ajouter s'il y a lieu un bit dans la partie entière (IWL) en observant la moyenne, l'écart type et la distribution des valeurs de chaque chemin de données par rapport à la valeur maximale que le chemin de données peut supporter.

iii) Augmenter la similitude entre les modèles virgule-flottante et virgule-fixe en resserant avec une certaine marge les critères de précision.

iv) Afin de s'assurer que chaque critère de précision sera respecté, effectuer une simulation avec un plus grand ensemble de données aléatoires une fois qu'une solution est trouvée.

En considérant ces stratégies, la méthode permet de trouver rapidement des solutions précises, tout en diminuant le temps de conception, le coût d'implantation, la consommation de puissance et en permettant l'augmention des performances.

Cette méthode peut-être appliquée à une grande variété d'algorithmes de traitement de signaux et dans le prochain chapitre, son utilitée est démontrée lorsque celle-ci est appliquée aux algorithmes de traitement d'images.

CHAPITRE 2

APPLICATION DE LA MÉTHODE AUX ALGORITHMES DE TRAITEMENT VIDÉO

La méthode proposée au chapitre précédent a été implantée et appliquée avec succès sur différents types d'algorithme de traitement de signaux dont: le filtre elliptic du 5^e ordre [24], le réseau de neurones Fuzzy Art [17], l'algorithme CORDIC [60], un décodeur itératif [16], la transformée discrète de cosinus [47] et son inverse [47], l'algorithme ELA de dé-entrelacement des images [40], des filtres graphiques (SUSAN [55], Hybride [45], Weiner [43]) et un égalisateur linéaire transversal [36]. Ce chapitre présente une platforme publiée dans [9], qui utilise la méthode afin d'analyser les algorithmes de traitement vidéo.

Cette plate-forme qui est illustré à la Figure 2.1, est composé de six modules: (1) un outil qui détermine automatiquement la taille des chemins de données (AWLDT), (2) un générateur de bruit, (3) un estimateur de la qualité des images, (4) un algorithme de traitement vidéo, (5) un pilote, et (6) un contrôleur.

Le premier module, AWLDT, est essentiellement la méthode proposée au chapitre précédent. La méthode détermine la résolution minimale que doivent prendre les opérandes contenus dans un algorithme de traitement vidéo, et en s'appuyant sur la résolution obtenue, la méthode détermine le coût d'implantation matériel de l'algorithme.

La tâche du second module consiste à incorporer un certain type et niveau de bruit dans des images provenant d'une bibliothèque. Quatre types de bruits ont été implantés dans la plate-forme: un bruit de compression JPEG, un bruit sel et

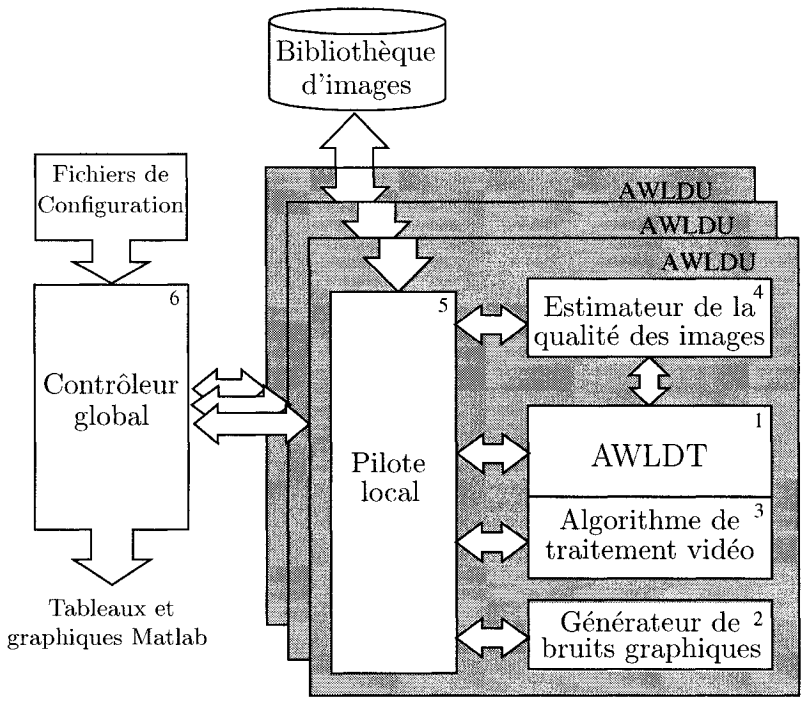


Figure 2.1 Plate-forme d'analyse des algorithmes de traitement vidéo

poivre, un bruit Gaussien et un bruit hors focus.

Le troisième module est l'endroit où l'algorithme de traitement vidéo est connecté au reste de la plate-forme. Il traite des images bruités par le second module, et génère des images restaurées.

Le quatrième module mesure la qualité Q des images restaurées, produites par l'algorithme de traitement vidéo. Il retourne la qualité résultante au pilote et à l'outil de détermination automatique.

Dans la plate-forme, l'outil de détermination automatique, l'algorithme de traitement vidéo, le générateur de bruit, le pilote et l'estimateur de la qualité des images forment une unité de calcul nommée AWLDU. Chaque unité est responsable d'effectuer une certaine tâche exigée par le contrôleur. Le contrôleur a la possibilité de démarrer l'exécution de plusieurs AWLDU simultanément. La tâche du pilote

consiste à gérer (a) les communications entre le contrôleur et l'AWLDU, et (b) les communications entre les différents modules du AWLDU. Au départ de l'exécution de l'AWLDU, le pilote reçoit du contrôleur le nom d'une image, un type de bruit, une séquence de niveau de bruit, et des objectifs de qualité. Par la suite le pilote distribue des commandes aux autres modules de l'unité AWDLU afin d'analyser l'algorithme de traitement vidéo. Finalement, lorsque le traitement est terminé, le pilote transmet au contrôleur les résultats de l'analyse pour les différents niveaux de bruit et objectifs de qualité.

Afin de déterminer adéquatement la résolution des opérandes et pour réduire la sensibilité de l'AWLDT aux entrées, l'analyse de l'algorithme de traitement vidéo devrait s'effectuer sur plusieurs images. Selon les caractéristiques du domaine d'application, il peut arriver que l'analyse de l'algorithme doit s'effectuée sur plusieurs types et niveaux de bruits. Sachant que l'analyse avec une seule image, un type de bruit et un niveau de bruit peut être très gourmande en temps de calcul, la plate-forme a été conçu afin que le contrôleur puisse superviser l'exécution de plusieurs AWLDU simultanément sur plusieurs ordinateurs. La tâche du contrôleur consiste donc à (i) démarrer sur les ordinateurs disponibles l'exécution d'une unité de calcul AWLDU, (ii) transmettre à chaque AWLDU le nom de l'image, le type de bruit, les niveaux de bruit et les objectifs de qualité à utiliser, (iii) récupérer les résultats de chaque AWLDU et (iv) compiler tous les résultats obtenus sous forme de graphiques et tableaux.

L'article présente le fonctionnement de la plate-forme lorsque l'analyse de l'algorithme SUSAN [55], un algorithme réducteur de bruit, est analysé. La Figure 2.2 illustre un exemple des résultats produits par la plate-forme, lorsque les images [65] traitées par algorithme sont affectées par un bruit Gaussien de niveau $\sigma = 65$. Trois courbes sont présentées dans cette figure: a) la qualité des images bruitées, b) la qualité des images restaurées lorsque les opérandes de l'algorithme

SUSAN ont une résolution vigule flottante IEEE à simple précision, et c) le coût d'implantation matériel de l'algorithme SUSAN en fonction de la qualité des images restaurées lorsque les opérandes ont une résolution virgule fixe.

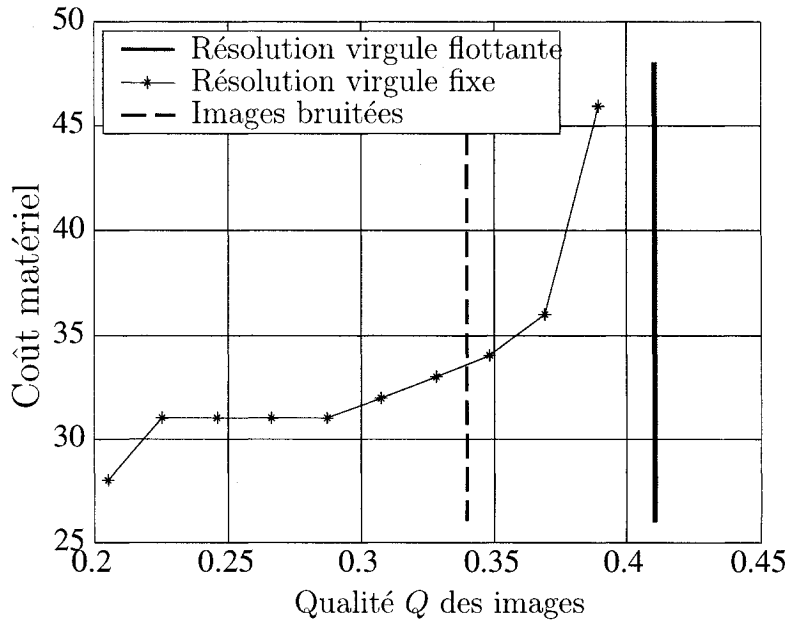


Figure 2.2 Coût d'implantation matériel de l'algorithme SUSAN en fonction de la qualité des images

Il est possible d'observer généralement une augmentation du coût d'implantation matériel de l'algorithme SUSAN au fur et à mesure que la qualité des images traitées augmente, et ce quelque soit le type et le niveau de bruit. Il est aussi possible d'observer qu'en dessous d'un certain coût matériel (un coût de 34 dans l'exemple de la Figure 2.2), l'algorithme SUSAN dégrade la qualité des images au lieu de l'augmenter.

Les temps nécessaires pour effectuer les différentes analyses de l'algorithme SUSAN sont présentés au Tableau 2.1. Le Tableau 2.1 reporte aussi l'accélération obtenue lorsque les analyses sont distribuées sur plusieurs ordinateurs. À partir de ce tableau, il est possible d'observer que l'accélération n'augmente pas linéairement avec le nombre d'ordinateurs et qu'une accélération de 12.27 est obtenue lorsque

l'analyse est distribuée sur 15 ordinateurs.

Tableau 2.1 Facteur d'accélération en fonction du nombre d'ordinateurs

Nombre d'ordinateurs	Nb AWLDU executions	Temps d'exécution (sec)	Accélération
15	56	2695.66	12.01
9	56	4220.77	7.67
5	56	8244.07	3.93
2	56	33074.41	0.98
1	56	32386.00	1.00
15	28	1529.30	10.02
9	28	2272.64	6.74
5	28	3734.99	4.10
2	28	15687.47	0.98
1	28	15323.74	1.00
15	14	612.44	12.27
9	14	1234.62	6.08
5	14	1978.66	3.79
2	14	7664.31	0.98
1	14	7512.31	1.00

Noter que le Tableau 2.1 est divisé en trois sous-ensembles correspondants au nombre d'exécutions AWLDU. Pour chaque sous-ensemble, l'accélération a été normalisée selon le temps nécessaire pour un seul ordinateur. Par conséquent, les accélérations des différents sous-ensembles ne peuvent pas être comparées entre elles.

La suite de ce chapitre présente l'article intitulé "A Unified Environment to Assess Image Quality in Video Processing" parut dans *Journal of Circuits, Systems, and Computers*, qui décrit plus en détails la plate-forme et l'analyse de l'algorithme SUSAN.

2.1 Article: A Unified Environment to Assess Image Quality in Video Processing

M.-A. Cantin, S. Regimbal, S. Catudal and Y. Savaria.
Electrical Engineering Department, École Polytechnique de Montréal,
C.P. 6079, succursale Centre-ville, Montréal, (Québec), Canada, H3C 3A7.

A new unified environment to analyze hardware implementations of video processing and noise reduction algorithms is proposed and analyzed. Based on an automatic word length determination tool, it evaluates the implementation costs required to reach a given quality target by performing several tests with noisy images at different noise levels. The unified environment uses a universal image quality index to compute the effect of finite precision on image quality. Also, this unified environment has the capacity to distribute the analysis task over a computer network in order to reduce the required processing latency. This unified environment helps to determine which algorithm requires the lowest implementation cost. Furthermore, the unified environment is useful to explore multiple hardware architectures that can be used to implement a given noise reduction algorithm.

2.1.1 Introduction

Noise can be introduced in digital images during the capture process or when a compression algorithm is applied on them to reduce their size for transport or storage purposes. Therefore, several techniques such as the box filter [1], the SUSAN noise reduction algorithm [2] and the Kalman filters [3] were elaborated and refined to remove noise found in these images.

In spite of very significant improvements in processor performance and power effi-

ciency, the requirements of many real-time image processing applications in term of pure performance or low power operation command the use of specialized hardware. However some noise reduction algorithms lead to high hardware and power consumption costs. Since cost, power dissipation, and performance are highly dependent on the number of bits used to represent data, and on the use of floating-point operators, the problem of precise word length determination is important. In some complex designs, half of the design time can be spent determining word lengths [4]. Many algorithms require a careful selection of word lengths to preserve their stability [5]. Therefore, an automatic word-length determination method is desired. In this paper, a unified environment to analyze the hardware implementation of noise reduction algorithms based on an automatic word length determination tool is presented.

In the next section, the unified environment that explores video processing and noise reduction algorithms to assess the resulting image quality is presented. The unified environment was implemented and is described in Section 2.1.3. In Section 2.1.4, the hardware implementation of a noise reduction algorithm is analyzed and the results obtained with the proposed method are reported and analyzed. Finally, our main conclusions are summarized in Section 2.1.5.

2.1.2 A Unified Environment

The unified environment, in which hardware implementations of noise reduction or video processing algorithms are analyzed, is composed of six major modules: the Automatic Word Length Determination Tool (AWLDT), the noise generation functions, the video processing or noise reduction algorithm, the image quality index estimator, the driver, and the environment controller (see Figure 2.3).

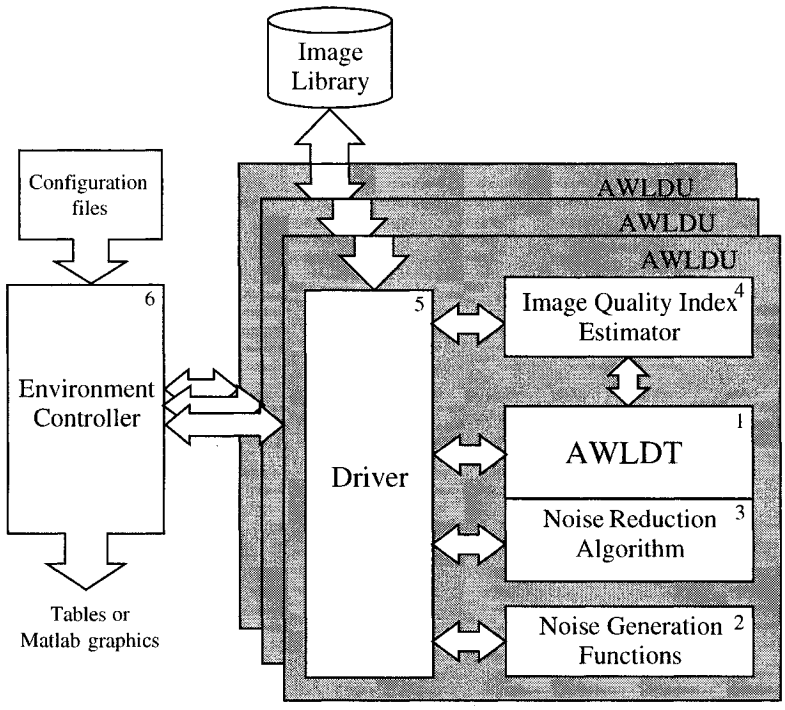


Figure 2.3 Unified Environment to Assess an Image Quality.

The first module is the AWLDT [6]. This module analyzes hardware implementations of noise reduction or video processing algorithms described in the C/C++ language. It finds the required bit resolution of each considered operand in the algorithm for a hardware architecture implementing it. In the last 10 years, three techniques have been developed to evaluate the word length of operands in general Digital Signal Processing (DSP) algorithm [7]. The AWLDT was selected because it uses a simulation-based approach known to be the most accurate technique that does not overestimate word lengths. This simulation based approach investigates quantization effects of finite word length between the fixed- and floating-point models through simulations of a testbench. Also, this technique can analyze DSP algorithms that contain data dependencies and does not need a fixed-point specification of input signals. However, it is time-consuming and may be input dependent.

A key feature of AWLDT is that the C/C++ description of the algorithm requires

a minimum number of modifications before analysis. Besides including header files, changing the name of the main function, and changing the declaration type of the operands analyzed, no other part of the algorithm needs to be changed. Fixed-point simulations instead of floating-point simulations are conducted automatically due to the operator overloading feature of the C++ language. In AWLDT, two fixed point simulation tools are available: SystemC [8] and the fixed point utility of W. Sung [9].

The second module is where noise generation functions are implemented. The noise generation functions used reflect the type of noise observed in many applications. They typically accept an argument that reflects the level of noise imposed on the image and they return a noisy image.

The third module is where the processing algorithm is connected to the rest of the environment. It inputs noisy images and returns the corresponding restored images.

The fourth module computes the image quality index based on the similarity between two images. It allows to quantify the performance of noise reduction algorithms and of quantization effects due to finite word lengths of operands in the algorithm. Finally the image quality index helps to set objectives and guides the automatic word length determination method towards an optimum hardware implementation.

The fifth module is the driver described lower. In the unified environment the AWLDT, the noise generation functions, the noise reduction algorithm, the image quality index estimation and the driver modules make up an Automatic Word Length Determination Unit (AWLDU). The AWLDU can be called several times by the environment controller. The driver acts as an interface between the environ-

ment controller and the other modules of the AWLDU. It receives an image and a specific noise sequence from the environment controller, and uses the other modules of the AWLDU to analyze the algorithm. Finally, results are retransmitted to the environment controller.

Finally, the last module is the environment controller. The unified environment was developed to compute the proper number of bits of a given noise reduction algorithm in order to reach an objective image quality. In order to perform an accurate word length determination of the algorithm, and to reduce the input sensitivity of the AWLDT, each word length determination must be performed on several images. Depending on the application, this word length determination could also be achieved for several noise types, noise levels and image quality objectives. The combination of a set of objectives, a noise level, an image and a noise type is considered as an evaluation. For each AWLDU execution, one evaluation is performed, thus several AWLDU executions must be performed. Hence, an efficient method is required to define the parameters of the different evaluations and also obtain the fastest possible execution. On the other hand, since a single execution of an AWLDU is time-consuming, the analysis of an algorithm may take a lot of time. The system that we propose to control the environment is designed to supervise several AWLDU executions. Hence, the system includes the determination of the parameters of each AWLDU execution. Then, it collects the results from each execution and produces comprehensive reports. Moreover, the system can distribute different AWLDU executions on a processor farm over a network. This improves the time required to evaluate an algorithm by leveraging available parallelism in the system.

2.1.2.1 Algorithm Evaluation Procedure

The unified environment to assess processed image quality image performs eleven execution steps.

1. In the first step, the controller reads the configuration file that contains a list of computers, a list of images, and a list of evaluations to perform.

2. In the second step, the controller starts a new execution of the AWLDU on each computer available in the list, and sends to each of them an evaluation to accomplish. As previously mentioned, to perform an evaluation, an AWLDU needs: the name of the image to process, a type of noise to inject with a noise level parameter specified, and a list of objectives. Note here that several types of noise with different noise levels could be superposed. The environment controller is responsible to allocate a specific evaluation for each available computer. When all computers of a network are allocated, the environment controller waits for the completion of an AWLDU execution to allocate another one to the available processor.

The environment controller is the core of the system. Figure 2.4 illustrates how the environment controller interacts with the other parts of the system.

3. For each execution of the AWLDU, the driver reads and memorizes the original image to process. Then the driver sends to the noise generation module the original image and the type of noise to inject with the noise level specified for the current evaluation.

4. The noise function module inserts noise and returns a noisy image.

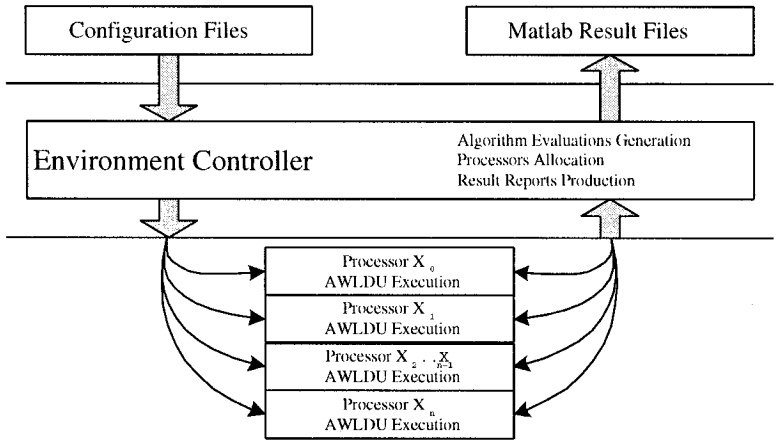


Figure 2.4 Environment Controller.

5. Once the noisy image is generated, the driver sends the noisy image to the noise reduction algorithm that tries to remove noise. The noise reduction algorithm returns a restored image.

6. The driver sends the original and restored images to the image quality index module in order to quantify the performance of the noise reduction algorithm in floating-point precision.

7. Once the image quality of the restored image processed in floating-point is computed, the image quality value is used as a reference to set image quality objectives. A quality objective could be set for example to 95 percent of the image quality produced by the noise reduction algorithm processed in floating-point precision.

8. For each quality objective, the driver sends to the AWLDT the original and noisy images and the quality objective to meet.

9. The AWLDT finds the optimum word length for each operand analyzed in the noise reduction algorithm that allows meeting the quality objective. During this step, the image quality index module is called many times to determine the quality achieved with a combination of word lengths with the objective

of finding the optimum word lengths combination. The implementation costs and the image quality resulting of the optimum word length combination are returned and temporarily accumulated in the driver block.

10. Step 9 is repeated until all quality objectives in the evaluation are reached. When an evaluation is completed, the results that were temporarily stored in the driver are returned to the environment controller and the AWLDU is released.

11. Steps 2 to 10 are repeated until all considered images are processed for every noise level and noise type specified in the configuration file. Finally, the environment controller creates comprehensive reports from all evaluations performed with the AWDLUs. These reports are generated using the MatLab™ format to enable automatic creation of graphs from the results.

These graphs help to determine which noise reduction or video processing algorithms require the smallest implementation costs. Furthermore, the unified environment is useful to explore multiple hardware architectures that can apply a same processing algorithm.

In the next section, the hardware implementation of a noise reduction algorithm found in the literature is described [2].

2.1.3 Implementation of the Unified Environment

The unified environment was implemented and applied to a video noise reduction algorithm. The next sub-sections will describe the details on how the test case reported in Section 2.1.4 was implemented.

2.1.3.1 Image set

The set of images considered in our analysis is part of the USC-SIPI[a] image database that is a collection of reference digitized images maintained primarily to support research in image processing, image analysis, and machine vision [10]. Specifically, the set of images considered in the environment consists of 14 gray-scale images (256 levels) composed of 128 x 128 pixels.

2.1.3.2 Types of Noise

Several types of noise were implemented in the environment to observe the performance of the noise reduction algorithm for each of them. These noise types were selected because they are commonly found in image processing applications. We present a brief description of each type of noise considered in our analysis. Moreover, the severity associated with each type of noise is reviewed.
The types of noise considered in our analysis are:

- Gaussian Noise

  Gaussian noise is often inserted in images during an analog transmission disturbed by random noise or electromagnetic interferences. We generate random noise values using the Box-Muller method [11] described by Eq. (2.1). The noise value is added to each pixel in the image.

$$n = \sigma\sqrt{-2\ln a}\cos(2\pi b), \tag{2.1}$$

  where a and b are two independent uniform random variables in the range $]0,1[$. Since the $\sqrt{-2\ln a}\cos(2\pi b)$ quantity has a zero mean and a standard

[a]University of Southern California-Signal and Image Processing Institute

deviation equal to 1, the severity of this noise is determined by the value of σ. When adding Gaussian noise to a pixel, the value of this pixel could exceed the minimum and maximum range of the image. Thus the value of the resulting pixel is saturated to the minimum or maximum bounds values allowed. For this type of noise, we analyzed deterioration using σ values from 5 to 65 with a step of 5.

- Salt and pepper

  The salt and pepper noise appears as randomly distributed black and white dots on the image. It is often caused by the capture process or by the means used to threshold an image. The severity of this noise is determined by the fraction of pixels, expressed in percent, affected by the algorithm. The noise injection levels considered in our analysis start from 5 to 65 percent with a step of 5 percent.

- JPEG Compression

  JPEG compression noise appears in images when the JPEG compression algorithm is applied on them. We used the standard JPEG compression tool in order to create realistic JPEG deterioration [12]. Then, using the standard JPEG decompression tool, we obtain noisy images. The severity of this noise is directly linked to the compression quality level used to compress an image [13]. Higher is the compression quality level, lower is the compression performed, and the noise injected into the images, where no compression refers to a compression quality level of 100. The compression quality levels considered in our analysis start from 5 to 50 with a step of 5.

- Blurring

  Blur noise appears in images when they are captured out of focus. It is often introduced in images through the capture or digitalization processes. The blur noise is generated by a low pass filter and a downsampling of the

image, which together is known as the Gaussian pyramid [14]. The Gaussian pyramid is defined as follows:

$$x_f(i,j) = \sum_{g=-2}^{2} \sum_{h=-2}^{2} w(g,h)\, x_{f-1}(2i+g, 2j+h) \tag{2.2}$$

where

$$w(g,h) = \begin{bmatrix} 0.0025 & 0.0125 & 0.0200 & 0.0125 & 0.0025 \\ 0.0125 & 0.0625 & 0.1000 & 0.0625 & 0.0125 \\ 0.0200 & 0.1000 & 0.1600 & 0.1000 & 0.0200 \\ 0.0125 & 0.0625 & 0.1000 & 0.0625 & 0.0125 \\ 0.0025 & 0.0125 & 0.0200 & 0.0125 & 0.0025 \end{bmatrix}$$

and where f is the level of the Gaussian pyramid, $x_0(i,j)$ is the original image, and $x_f(i,j)$ is the resulting image of the f^{th} level. In the Gaussian pyramid, higher is the level, higher is the severity of the blur noise. When applying the Gaussian pyramid to an image, the number of pixels of the resulting image is reduced by a multiplicative factor of 2^{f-1}. The number of pixels is restored by expanding the resulting image as follows:

$$y_f(i,j) = 4 \sum_{g=-2}^{2} \sum_{h=-2}^{2} w(m,n)\, y_{f+1}\left(\frac{i-g}{2}, \frac{j-n}{2}\right) \tag{2.3}$$

where $y_f(i,j)$ is the expanded image of the f^{th} level, and where y_0 is the resulting noisy image with the same size as the original image x_0. In our analysis, levels 1 to 5 were considered.

For every image considered in our analysis, the implementation cost of the noise reduction algorithm is found for several noise types and severities.

2.1.3.3 Image Quality Index

There are three possible situations in which an image quality metric could be needed. These situations are:

1. to monitor image quality for quality control systems
2. to benchmark image processing algorithms
3. to embed an image quality metric into image processing systems that automatically optimizes algorithms and parameter settings.

In our case, the motivation to use an image quality metric comes from the second situation. Since our primary goal is to remove human observers from the process of quality measurement, we need to use an objective quality metric that is able to make a consistent difference between an original and a distorted image. Image distortion assessment are divided in two distinct groups. The first group refers to mathematically defined measures such as mean square error, correlation quality, signal to noise ratio, etc [15]. The second group makes measurement considering Human Visual System (HVS) characteristics in an attempt to incorporate perceptual quality measurement [16].

An existing algorithm named the universal image quality index (UIQI) provides a metric independent from images being tested and viewing conditions [17]. The other interesting factor when using this algorithm is the fact that it has also a very low computational complexity compared to HVS models. The UIQI is defined as

$$Q = \frac{4\sigma_{xy}\overline{x}\ \overline{y}}{\left(\sigma_x^2 + \sigma_y^2\right)\left[(\overline{x})^2 + (\overline{y})^2\right]}, \tag{2.4}$$

where

$$\overline{x} = \frac{1}{N}\sum_{i=1}^{N} x_i, \quad \overline{y} = \frac{1}{N}\sum_{i=1}^{N} y_i,$$

$$\sigma_x^2 = \frac{1}{N-1}\sum_{i=1}^{N} (x_i - \overline{x})^2, \quad \sigma_y^2 = \frac{1}{N-1}\sum_{i=1}^{N} (y_i - \overline{y})^2,$$

$$\sigma_{xy} = \frac{1}{N-1}\sum_{i=1}^{N} (x_i - \overline{x})(y_i - \overline{y}),$$

where x is the original image signal, y is the noisy image signal, and where N is the number of pixels of a sliding windows [18]. The dynamic range of the universal image quality index Q is [-1,1] where 1 is the best value obtained when the two compared images are identical. The image quality is evaluated by combining three metrics as can be seen when Eq. (2.4) is rewritten as follows [17]:

$$Q = \frac{\sigma_{xy}}{\sigma_x \sigma_y} \cdot \frac{2\overline{x}\ \overline{y}}{(\overline{x})^2 + (\overline{y})^2} \cdot \frac{2\sigma_x \sigma_y}{\sigma_x^2 + \sigma_y^2} \tag{2.5}$$

From left to right in Eq. (2.5), the three terms are the degree of linear correlation, the similarity between the luminances, and the similarity between the contrasts of two considered images. Since an image is often space variant, the authors suggest to compute the quality index based on local quality indexes computed within a sliding window. Then, the quality index is equal to the mean of the local quality indexes (see Eq. (2.6)).

$$Q = \frac{1}{M}\sum_{m=1}^{M} Q_m. \tag{2.6}$$

where M is the number of different positions that could take the sliding window within the image and where Q_m is the local quality index of the m^{th} position. In the following, we use 6 by 6 pixels (N=36) windows in our analysis. That parameter can be changed as required by the user.

2.1.3.4 The Automatic Word Length Determination Tool

The AWLDT offers nine maximization procedures to find a word length combination that meets the image quality objective [7]. Each maximization procedure tries to find the optimal combination of word lengths that maximizes a metric C (see Eq. 2.7) in a number of iterations as small as possible. The metric C is defined as follows:

$$C = \left[\left(\frac{Q_{acq} - Q_{obj}}{Q_{obj}}\right) d_e\right] \cdot H + \left[\left(\frac{Q_{acq} - Q_{obj}}{Q_{obj}}\right) \cdot (1 - d_e)\right] \cdot \frac{1}{H} \qquad (2.7)$$

with

$$\begin{aligned} d_e &= 1 \text{ if } Q_{obj} \geq Q_{acq} \\ d_e &= 0 \text{ if } Q_{obj} < Q_{acq} \end{aligned},$$

where Q_{obj} is the image quality objective, Q_{acq} is the image quality obtained by the noise reduction algorithm for a specific word length combination and H denotes the implementation costs.

With this metric, all considered word length combinations are graded to find the optimal solution. For two word length combinations having the same implementation cost, the one that maximizes the image quality yields a higher value of the

metric. For two word length combinations producing the same image quality, the one that minimizes the implementation costs yield a higher value of the metric. The maximum value of the metric is obtained at the optimal solution. The metric generates a negative value if the image quality objective is not met. Otherwise the metric generates a positive value.

The implementation costs is defined as:

$$H = \mathcal{F}\left(WL_0, WL_1, \ldots, WL_{I-1}\right),$$

where I is the number of operands analyzed in the noise reduction algorithm, WL_i is the word length of the i^{th} operand, and $\mathcal{F}(\cdot)$ is an objective function that reflects any combination of performance, hardware or power consumption costs suitable for a specific platform. The AWLDT was enhanced to give users the possibility to specify desired implementation costs functions. Thus, if a user wants to analyze a hardware architecture on two different platforms, on an ASIC and an FPGA for example, he can only replace the objective function. In the following analysis, for simplicity, we use a linear relationship and set to 1 the cost per bit for all operands analyzed in the noise reduction algorithm. That can easily be generalized to more realistic cost functions as a function of the parameter to optimize (power or area for example) or implementation technology (ASIC or FPGA for example). Finally, based on previous experience, the heuristic procedure was chosen to find an optimum solution in a minimum number of iterations [7].

2.1.3.5 A Noise reduction Algorithm

Since 1980, several noise reduction methods were elaborated to remove the noise from images. These methods can be categorized into three groups: the linear filters, the non-linear filters, and the adaptive filters.

In the linear filters group, the simplest noise reduction method is the box filter [1]. In this method each pixel is replaced by the mean of its local neighbours. The Gaussian filter, which is the most widely used noise reduction method, is similar to the box filter method, except that the values of the neighbours pixels are given some weight defined by a spatial Gaussian distribution. These linear filters are fast but they tend to create more blurring in image details.

In the non-linear filters group, for each pixel, a list of the local neighbours sorted into ascending order, according to their values, is first created. Then operations are made on the list. The median [19], weight median [20] and midrange estimator [21] filters are three examples that can be cited over a large set of methods.

Finally, the third group tends to preserve the structure of images by making target operations on image regions depending of their characteristics.

The Kalman filters [3] and the Smallest Univalue Segment Assimilating Nucleaus (SUSAN) [2] noise reduction algorithms are two methods using this technique to reduce noise in images. The authors of the SUSAN algorithm said that: "... This filter clearly integrates the best aspects of the best existing noise reduction filters including edge preserving filters, ...". For this reason, and because it is a low level image processing method, the SUSAN noise reduction algorithm was implemented and analyzed in our unified environment. Let us stress that in the unified environment, any video processing or noise reduction algorithm can easily be implemented.

The complete equation used for the SUSAN filter is:

$$y'(i,j) = \frac{\sum_{k=0}^{K-1}\sum_{l=0}^{K-1} y(i+k,j+l)\cdot E(i,j,k,l)\Big|_{(i,j)\neq(0,0)}}{\sum_{k=0}^{K-1}\sum_{l=0}^{K-1} E(i,j,k,l)\Big|_{(i,j)\neq(0,0)}}, \tag{2.8}$$

where

$$E(i,j,k,l) = e^{-\frac{r^2}{2d^2} - \frac{(y(i+k,j+l)-y(i,j))}{t^2}},$$

and where $r = \sqrt{i^2+j^2}$, y is the noisy image, y' is the restored image, d control the scale of the spatial smoothing, t is the brightness threshold, and K is the width of the sliding window.

The analysis determines the bit resolution of the input pixels, the $\sum\sum E(i,j,k,l)$ quantity, the $\sum\sum y(i+k,j+l)\cdot E(i,j,k,l)$ quantity, and the bit resolution of the output pixels, with t set to 20, d set to 1 and K set to 4.

2.1.3.6 The Environment Controller

The environment controller is implemented using the *e* language. The *e* language is a specialized language generally used for hardware functional verification. It is a proprietary language commercialized by Verisity^{TM}. However, a Project Authorization Request, named IEEE 1647 has recently been approved by the IEEE Design Automation Standards Committee to develop a standard verification language based on the *e* language. Similarly to C++, *e* is a high level language. In addition, it provides several elements dedicated to hardware verification such

as a pseudo-random generator, a temporal algebra and functional coverage facilities [22]. An *e* program executes on a simulator named Specman Elite^{TM} We used the *e* language for two reasons. We want to create a generic environment that can be configured. Then, the language includes a library, the Co-Verification Link (CVL), that supports interactions between an *e* program and other software running on the same or on another computer. Since the *e* language is typically used for hardware verification, CVL was developed to support hardware/software co-verification. That functionality could be implemented in C but the CVL was quite effective in meeting our requirements for distributing several AWLDU executions over a computer network. The CVL uses operating system sockets to establish communication between an *e* program and external software. The environment takes as entry a configuration file. These files specify generation constraints defined using the format given by the *e* language.

2.1.4 Result and Analysis

The evaluation of the SUSAN algorithm aimed at finding the implementation costs for the images, noise types, and noise levels presented in the previous section. Furthermore, the evaluation tried to minimize the implementation costs necessary to reach image quality objectives starting from 50 to 95 percent of the one obtained in floating-point precision with a step of 5 percent.

The unified environment was run over 15 computers, where 1 computer, a 300MHz Utra-Sparc 10 station was used to run the environment controller and 14 450MHz SunBlade 110 stations were used to run 14 concurrent AWLDU executions.

Derived from this evaluation, Figures 2.5 to 2.10 show respectively the implementation costs when the original images are affected by σ equals to 5 and 65 for the

Gaussian noise, 5 and 65 percent of the salt and pepper noise, and a compression quality equal to a level of 5 and 50 for the JPEG compression. The implementation costs when the original images are affected by the blur noise are not shown because it was observed in the detailed simulations that the SUSAN algorithm cannot remove this type of noise. Furthermore the SUSAN algorithm deteriorates the image quality of the blurred images.

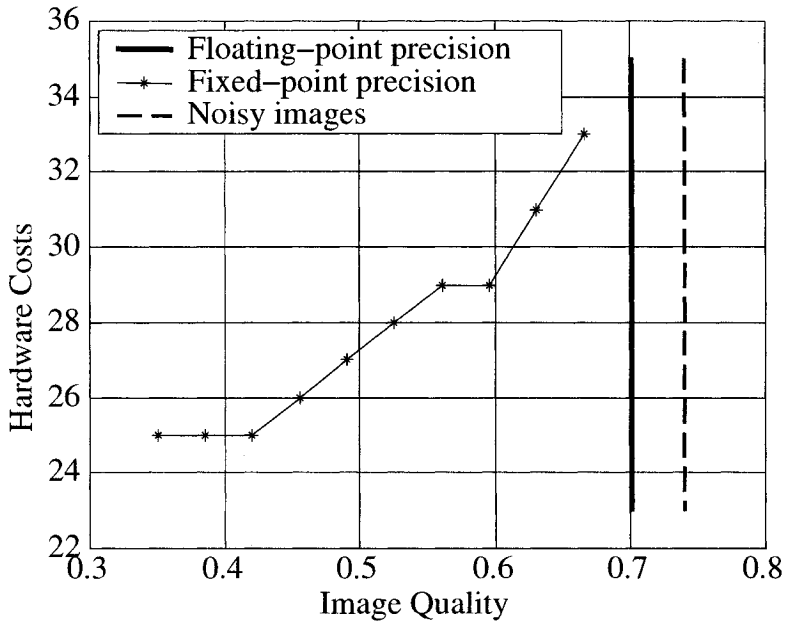


Figure 2.5 Gaussian noise with σ equals to 5

On each of these figures, 3 curves are presented. The vertical dashed curve shows the average image quality for the 14 noisy images. The vertical solid curve shows the average image quality for the 14 restored images produced by the SUSAN algorithm processed in floating-point precision. Finally, the last curve presents the implementation costs of the optimum word length combination found for different image quality objectives.

Figures 2.6, 2.8 and 2.9 show the image quality obtained with the Gaussian noise when σ equals 65, when 65 percent of salt and pepper noise is injected, and when JPEG compression is applied with a quality level of 5. The scores obtained with

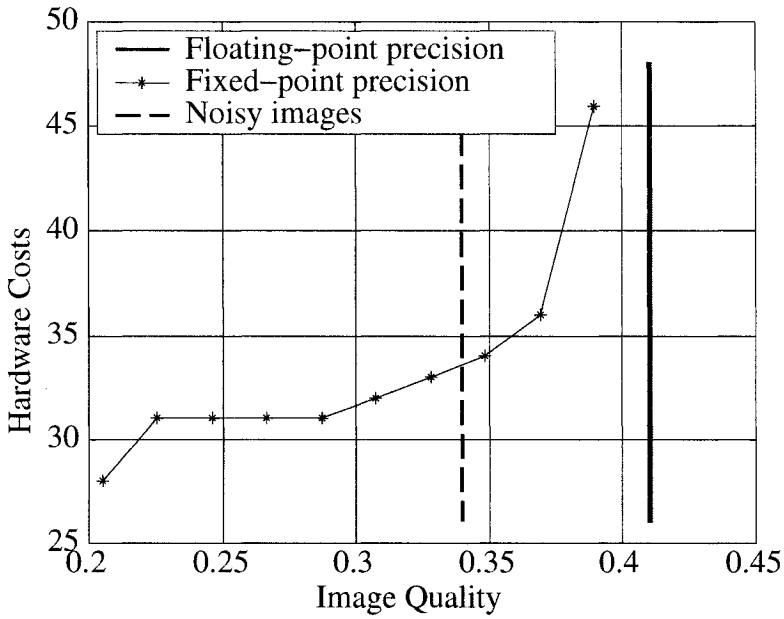


Figure 2.6 Gaussian noise with σ equals to 65

the image quality metric were 0.34, 0.45, and 0.40 respectively. When applying the SUSAN algorithm in floating-point precision, these image qualities are respectively restored to 0.41, 0.52, and 0.42. However, when the severity of these types of noise are low, as shown in Figures 2.5, 2.7, and 2.10, the average image quality of the images, obtained with the SUSAN algorithm processed in floating-point precision, is lower than the average image quality computed with the raw noisy images. These figures show that higher is the quality objective, higher must be the bit resolution to reach these quality, and thus higher are the implementations costs. In Figure 2.10, for some images, the AWLDT could not find any word length combination that produce an image quality higher than 0.65.

With these graphs, the degradation due to fixed-point processing is obvious. Also, it is easy to determine what is the implementation cost required to satisfy a target image quality. Let us stress that the tool does not only produce an evaluation of the implementation complexity, but also produces the exact word length that must be used for each operand in an implementation to reach these quality levels.

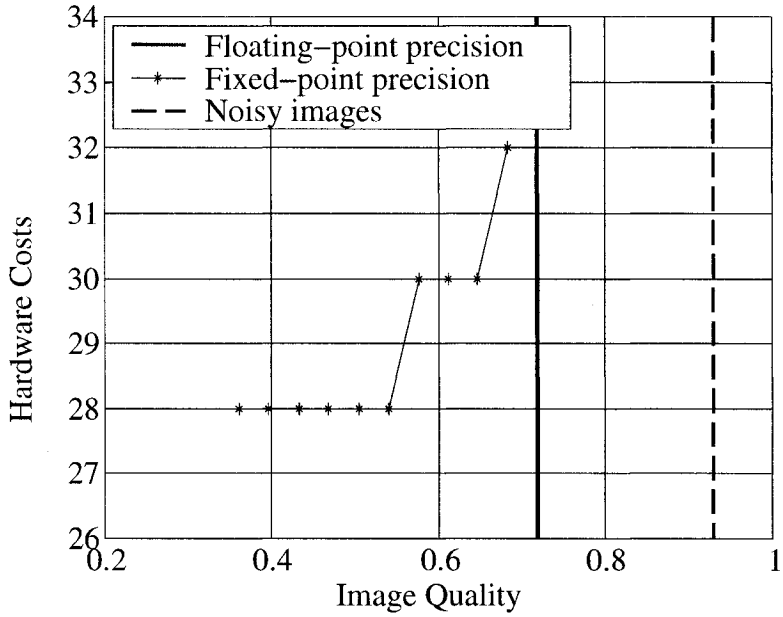


Figure 2.7 5 percent of salt and pepper

The time required to compute an evaluation when it is executed on 1, 2, 5, 9, and 15 computers is reported in Table 2.2. Furthermore, the times to compute evaluations requiring 14, 28 and 56 AWLDU executions are also reported. The number of AWLDU executions is a multiple of 14 because the image database contains 14 images. Since one AWLDU execution performs an analysis of one image and one noise level, Table 2.2 shows the time required to compute an evaluation that contains 1, 2, and 4 different noise levels. For each AWLDU execution, the AWLDT was called 10 times to find the implementation costs of 10 different image quality objectives.

For these simulations, it was decided to execute the environment controller on a separate computer when the evaluation is performed on more than one computer. This constraint allows the environment controller to interface rapidly with the AWLDU executions. Furthermore, it allows the environment controller to efficiently control a large number of AWLDU executions in parallel.

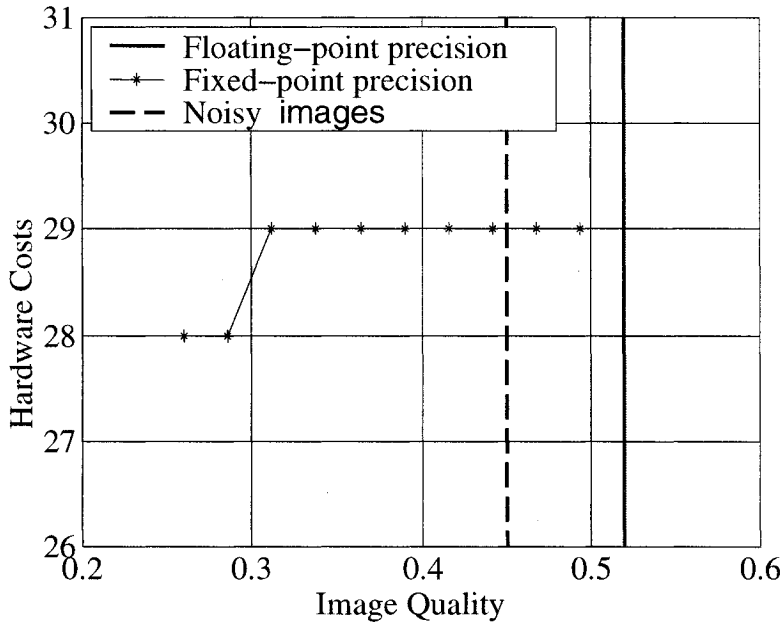


Figure 2.8 65 percent of salt and pepper

Finally, the acceleration obtained when the evaluation is distributed over 2, 5, 9, and 15 computer is reported on the last column of Table 2.2.

From these results, we first observe that the acceleration does not increase linearly with the number of computers. Furthermore, the time is increased by a factor of 1.02 when the evaluation is performed on 2 computers. This increase can be caused by the time overhead required for the communication between the two computers. However, when 5 or more computers are used, the execution is always faster than if it is executed on only one computer. Specifically, the maximum mean time required to compute one AWLDU execution on one computer is equal to 589.89 seconds. This time is reduced to 147.23, 88.19, and 54.62 seconds when respectively 5, 9, and 15 computers are used.

The fact that the acceleration obtained with 28 AWLDU executions on 15 computers differs up to 18 percents from the acceleration obtained with 14 and 56 AWLDU executions was attributed to the fact that the times reported in Table 2.2. are based

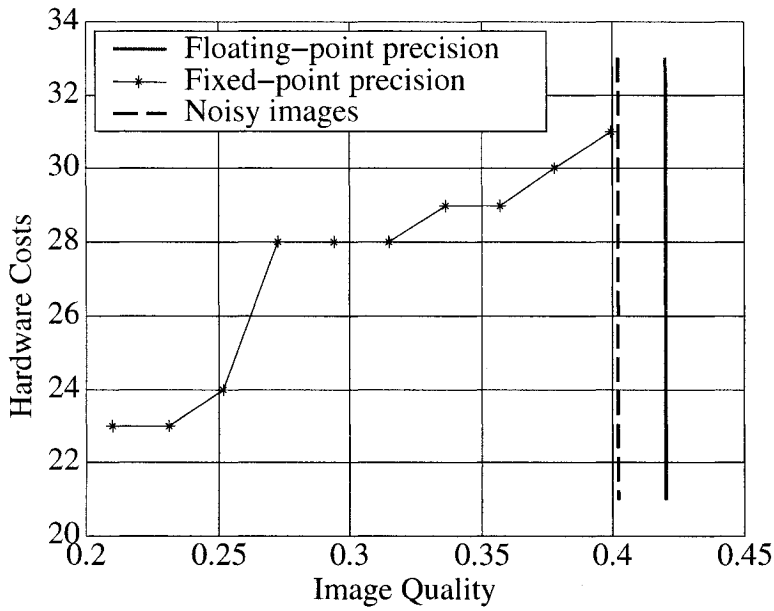


Figure 2.9 JPEG compression quality level of 5

on one simulation performed on a computer network that could be affected by other processes on the computers used.

Since the environment controller could not start two AWLDU executions simultaneously, when two AWDLU executions finish at the same time, during the time the environment controller starts a new AWDLU execution on the first computer, the second computer stays idle. Thus, we expect a saturation of the acceleration when the environment controller cannot manage efficiently a high number of computers. This saturation could not be experimentally observed, because the number of computers required to obtain this limit exceeds the number of computers available in our laboratory.

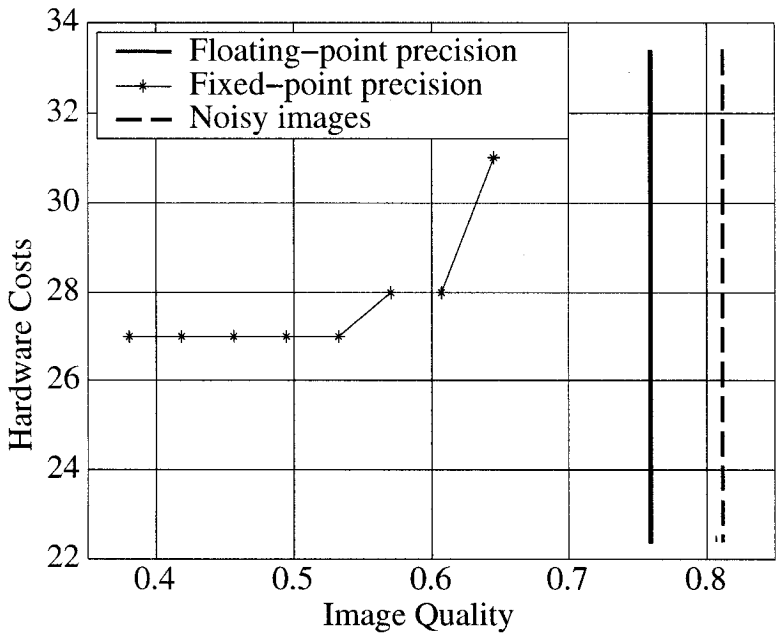


Figure 2.10 JPEG compression quality level of 50

2.1.5 Conclusion

An automated method supported by a unified environment was proposed to optimize hardware implementation of video processing algorithms that meet a target objective image quality. The unified environment uses an automatic word length determination tool that finds an optimum word length combination to meet an image quality objective. It also uses a Universal Image Quality Index to compute the effects of finite precision on processed image quality. The unified environment was implemented and a noise reduction algorithm was optimized to obtain the minimum cost solution that produces a target image quality. The environment supports four types of noise with parametrized noise levels.

It was found that the unified environment reduces the processing time by a factor of up to 12.27 when the task is distributed over a 15 computer network. Also, it reduces the input sensitivity of the automatic word length determination tool by

Tableau 2.2 Acceleration of the processing time vs Number of computers

Number of Computers	Nb AWLDU executions	CPU time (sec)	Acceleration
15	56	2695.66	12.01
9	56	4220.77	7.67
5	56	8244.07	3.93
2	56	33074.41	0.98
1	56	32386.00	1.00
15	28	1529.30	10.02
9	28	2272.64	6.74
5	28	3734.99	4.10
2	28	15687.47	0.98
1	28	15323.74	1.00
15	14	612.44	12.27
9	14	1234.62	6.08
5	14	1978.66	3.79
2	14	7664.31	0.98
1	14	7512.31	1.00

performing the determination over several images.

The unified environment helps to determine which video processing or noise reduction algorithms requires the lowest implementation costs. It also helps to determine actual implementation costs and it is useful to explore multiple hardware architectures that may target a range of implementation technologies or platforms.

Acknowledgments

The authors want to thank Micronet, and the Natural Sciences and Engineering Research Council of Canada for their financial support. This project was made possible by the donation of a license of the Specman Elite^{TM} software by Verisity.

References

[1] M.J. McDonnell. Box-filtering techniques. *Computer Graphics and Image Processing*, vol. 17:65–70, 1981.

[2] S.M. Smith and J.M. Brady. SUSAN - A New Approach to Low Level Image Processing. *International Journal of Computer Vision*, vol. 23:45–78, 1997.

[3] R. Kalman. A New Approach to Linear Filtering and Prediction Problems. *Transactions of the ASME, Journal of Basing Engineering*, vol. 82:34–35, 1960.

[4] H. Keding, M. Willems, M. Coors, and H. Meyr. FRIDGE: a fixed-point design and simulation environment. *Design, Automation and Test in Europe*, pages 429–435, 1998.

[5] M.-A. Cantin, Y. Blaquière, Y. Savaria, P. Lavoie, and E. Granger. Analysis of quantization effects in a digital hardware implementation of a fuzzy ART neural network algorithm. *IEEE Int. Symposium on Circuits and Systems*, vol. 3:141–144, 2000.

[6] M.-A. Cantin, Y. Savaria, D. Prodanos, and P. Lavoie. An automatic word length determination method. *IEEE International Symposium on Circuits and Systems*, vol. 5:53–56, 2001.

[7] M.-A. Cantin, Y. Savaria, and P. Lavoie. A Comparison of Automatic Word Length Optimization Procedures. *IEEE International Symposium on Circuits and Systems*, vol. 2:612–615, 2002.

[8] M. Speitel and B. Niemann. SystemC design language for development of ASICs and systems. *Elektronik*, vol. 50(13):78–83, 2001.

[9] S. Kim, Ki-Il Kum, and W. Sung. Fixed-Point Optimization Utility for C and C++ Based Digital Signal Processing Programs. *IEEE Transaction on Circuits and Systems II*, vol. 45(11):1455–1464, 1998.

[10] A. G. Weber. The usc-sipi image database: Version 5. Master's thesis, University of Soutern California, 1997.

[11] G.E.P. Box and M.E. Muller. A note on the generation of normal random deviates. *Annals of Mathmatical Statistics*, vol. 29:610–611, 1958.

[12] G. Wallace. The jpeg still picture compression standard. *Communications of the ACM*, vol. 34(4):30–44, 1991.

[13] Tom G. Lane. The independant jpeg group's software, release 5b, 1990-1995. Archive site: *ftp.uunet.*

[14] P.J. Burt and E.H. Adelson. The laplacian pyramid as a compact image code. *IEEE Transactions on Communications*, pages 532–540, 1983.

[15] A.M. Eskicioglu and P.S. Fisher. Image quality measures and their performance. *IEEE Transactions on Communications*, vol. 43(12):2959–2965, 1995.

[16] T.N. Pappas and R.J. Safranek. *Perceptual criteria for image quality evaluation.* Handbook of Image and Video Processing. Academic Press, 2000.

[17] Z. Wang and A.C. Bovik. A universal image quality index. *IEEE Signal Processing Letters*, vol. 9(3):81–84, 2002.

[18] Z. Wang and A.C. Bovik. Why is image quality assessment so difficult? *IEEE International Conference on Acoustics, Speech, and Signal Processing*, vol. 4:3313–3316, 2002.

[19] J.W. Tukey. *Exploratory Data Analysis.* Addison-Wesley, 1971.

[20] D.R.K. Brownrigg. The weighted median filter. *Communications of the ACM*, vol. 27:807–818, 1984.

[21] G.R. Arce and S.A. Fontana. On the midrange estimator. *IEEE Trans. on Acoustics, Speech and Signal Processing*, vol. 36:920–922, 1988.

[22] S. Palnitkar. *Design Verification with e.* 2002.

Tel que démontré dans l'article précédent, le temps de l'analyse a été réduit en distribuant la charge de calcul sur plusieurs ordinateur.

Une autre approche qui réduit la durée de l'analyse par la méthode est présentée dans [52]. Cette approche effectue l'optimisation des tailles des chemins de données d'un algorithme directement sur son implantation matériel. Cette approche a permis d'effectuer une optimisation d'un décodeur à seuils itératifs avec la méthode proposée au Chapitre 1 en 1.58 heures, qui autrement aurait nécessité 321.16 jours de simulation sur un Sparc V440.

Dans le prochain chapitre la méthode proposée est modifiée afin d'étendre son utilitée à d'autres types d'optimisations et d'applications.

CHAPITRE 3

EXTENSION DE LA MÉTHODE À D'AUTRES APPLICATIONS ET TYPES D'OPTIMISATIONS

La méthode proposée au Chapitre 1 a été adaptée afin de l'appliquer à deux autres types d'optimisation.

3.1 Optimisation des algorithmes itératifs

Tout d'abord, la méthode a été adaptée afin d'optimiser des algorithmes itératifs en déterminant simultanément la taille des chemins de données et le nombre de fois qu'une itération doit y être exécutée.

Un algorithme itératif est caractérisé par l'exécution successive d'une séquence d'opérations (appelée itération), afin que les résultats obtenus s'approchent graduellement de ceux escomptés. Cette séquence d'opérations est effectuée un certain nombre de fois, ou jusqu'à temps qu'une certaine condition soit rencontrée. Généralement, la première itération permet d'effectuer une première estimation des résultats et au fur et à mesure que le nombre d'itérations augmente, les résultats deviennent de plus en plus précis.

Cependant, lorsqu'un algorithme itératif est implanté avec une certaine combinaison de tailles de chemins de données, l'accroîssement du nombre d'itérations n'augmente pas toujours la précision des résultats. Lorsque la précision produite par une $n^{\text{ième}}$ itération est supérieure à la résolution fournie par la combinaison des tailles des chemins de données, l'ajout d'itérations additionnelles n'apportera pas

plus de précision dans les résultats. Dans cette situation, la précision des résultats ne peut augmenter que si la résolution de la combinaison des tailles des chemins de données est augmentée.

Il en découle que pour une certaine combinaison des tailles des chemins de données, il existe un nombre optimal d'itérations, et pour un certain nombre d'itérations, il existe une combinaison optimale des tailles des chemins de données.

Par conséquent, il peut être avantageux d'optimiser simultanément le nombre d'itérations et les tailles des chemins de données dans un même processus d'optimisation afin d'obtenir une solution qui rencontre les spécifications de l'usager.

Par analyse, il est possible d'observer à la Figure 3.1 que le nombre d'itérations produit un comportement très similaire à celui produit par la taille d'un chemin de données. De façon générale, l'augmentation du nombre d'itérations tend à augmenter la précision des résultats et lorsque le nombre d'itérations est réduit, la précision des résultats est réduite.

Sur la base de cette observation, la méthode présentée au Chapitre 1 a été adaptée afin de déterminer simultanément la taille des chemins de données et le nombre d'itérations en réalisant les modifications suivantes:

1. Création d'une nouvelle classe C++ qui spécifie quelles sont les variables qui déterminent le nombre d'itérations.
2. Lors d'une simulation en virgule-flottante de l'algorithme itératif, les variables qui déterminent le nombre d'itérations sont fixées égales à $WL^{\max}$ (la résolution maximale supportée par le simulateur virgule-fixe).
3. La méthode fixe les valeurs des variables qui déterminent le nombre

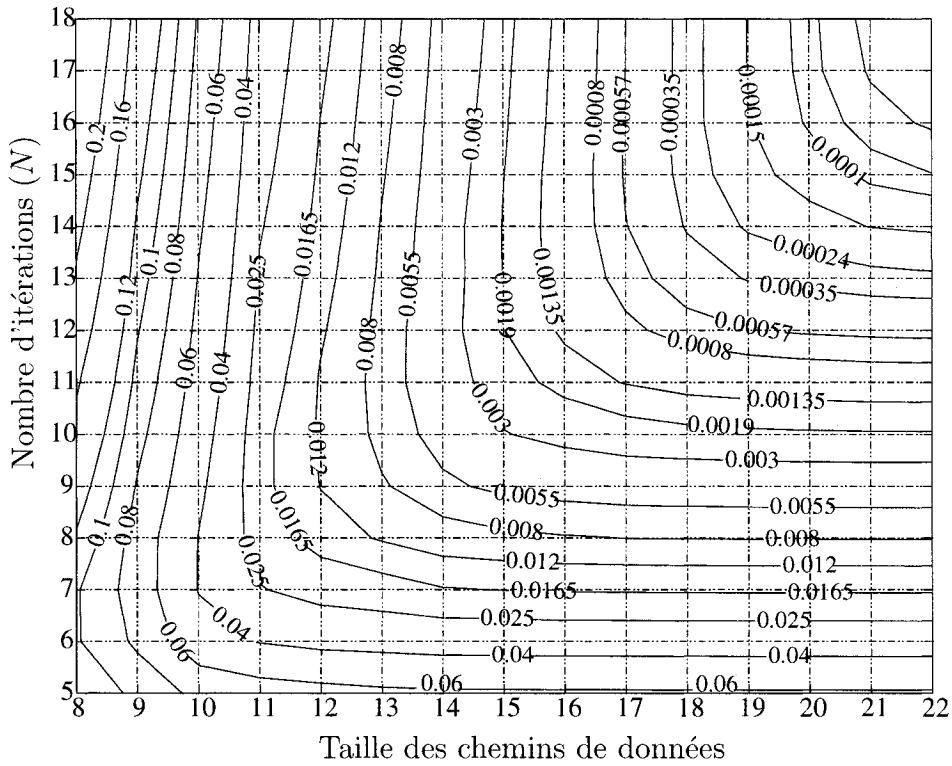


Figure 3.1 Erreur maximum de l'algorithme CORDIC en fonction du nombre d'itérations et de la taille des chemins de données

d'itérations à leur déclaration. En aucun moment, l'usager ne peut assigner une valeur à ces variables.

Cette nouvelle méthode a été implantée. Elle a permis de déterminer avec succès la taille des chemins de données et le nombre d'itérations de plusieurs algorithmes itératifs, dont l'algorithme CORDIC [60], l'estimateur fréquentiel de Crozier [26], un décodeur à seuil itératif [16] et l'algorithme de Newton [44].

Les résultats obtenus avec cette méthode pour l'algorithme CORDIC, sont présentés dans [12]. Les résultats présentés sont comparés à ceux obtenus par un modèle analytique présenté par Kota dans [37]. Ce modèle spécifie que pour

obtenir N bits de précision dans les résultats, l'algorithme CORDIC doit effectuer N itérations et les chemins de données doivent comporter $N+\log_2 N+2$ bits lorsque ceux-ci sont tous restreints à la même taille. Par exemple, pour obtenir 8 bits de précision dans les résultats, l'algorithme CORDIC doit effectuer 8 itérations avec des chemins de données de 11 bits. Cependant, contrairement au modèle de Kota, il est démontré dans [12], que pour différentes architectures, il existe différentes solutions d'implanter l'algorithme CORDIC.

Hampson [29] présente deux architectures de base, l'une sérielle et l'autre parallèle, pour implanter l'algorithme CORDIC. Tel qu'illustré à la Figure 3.2, l'architecture sérielle est composée de 3 additionneurs-soustracteurs, 3 registres, 2 décaleurs, 3 multiplexeurs, 1 compteur et une mémoire LUT. Cette architecture permet de calculer un résultat à tous les N cycles, où N est le nombre d'itérations exécutées dans l'algorithme CORDIC.

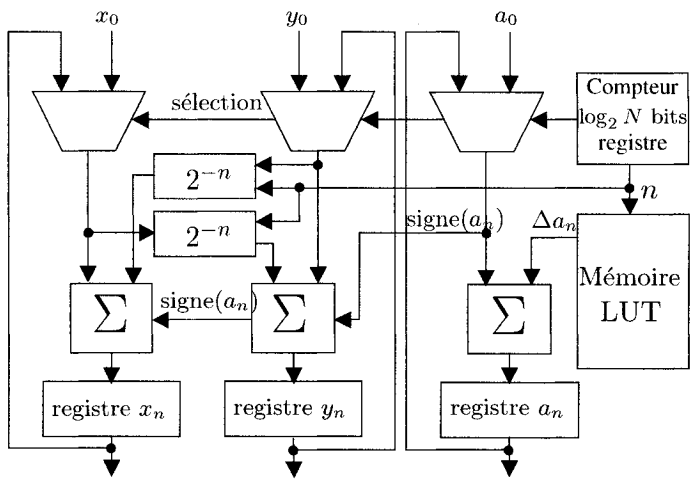


Figure 3.2 Architecture sérielle pour exécuter l'algorithme CORDIC

L'architecture parallèle, tel qu'illustrée à la Figure 3.3, est composée de $3 \cdot N$ registres et additionneurs-soustracteurs. Comme l'architecture sérielle, elle produit un résultat à tous les N cycles. Cependant, l'architecture parallèle permet de produire un résultat à chaque nouveau cycle d'horloge.

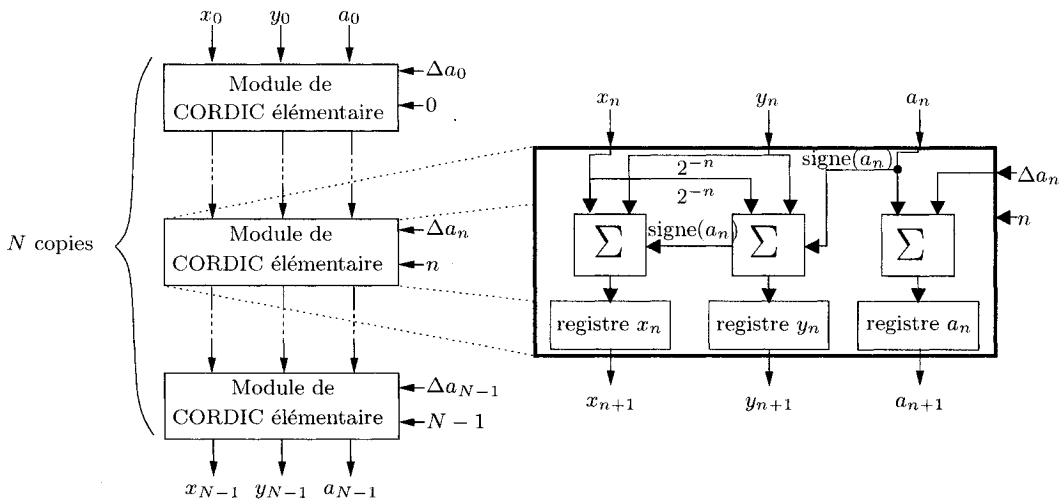


Figure 3.3 Architecture parallèle pour exécuter l'algorithme CORDIC

Le Tableau 3.1 présente les résultats obtenus avec la méthode proposée pour plusieurs critères de précision. Ces résultats ont été obtenus lorsque les tailles des chemins de données WL_a, WL_x, WL_y, WL_{Δ_a}, WL_{Δ_x}, et WL_{Δ_y} sont fixées au même nombre de bits, afin de pouvoir comparer ces résultats avec ceux obtenus par le modèle analytique de Kota [37]. Finalement, ces résultats ont été obtenus lorsque le coût d'implantation H de l'architecture sérielle, est calculée selon l'équation 3.1 et lorsque le coût d'implantation de l'architecture parallèle est calculé selon l'équation 3.2. Ces deux équations ont été obtenues par analyse des Figures 3.2 et 3.3 lorsque le coût matériel associé à un bit d'un registre, d'un additionneur, d'un multiplexeur, d'un compteur, d'un décaleur et d'une mémoire, sont tous fixés à 1.

$$H = ((3 \cdot WL_x) + WL_{\Delta x} + (3 \cdot WL_y) + WL_{\Delta y} + (3 \cdot WL_a)) + (2 \cdot \lceil \log_2(N) \rceil) + (WL_{\Delta a} \cdot N) \tag{3.1}$$

$$H = N \cdot ((2 \cdot WL_x) + (2 \cdot WL_y) + (2 \cdot WL_a)) \tag{3.2}$$

Tableau 3.1 Solutions obtenues par la méthode de détermination automatique

Arch.	S_0	Précision Obtenue ε_0	WL_x, WL_y, and WL_a	N	H	Métrique C	Nombre de simulations
serial	0.040000	0.039272	10	7	186	0.338808	95
parallel	0.040000	0.039069	11	6	396	0.159150	95
serial	0.025000	0.019854	11	8	215	0.293981	91
parallel	0.025000	0.020764	12	7	504	0.125336	91
serial	0.008000	0.007039	13	10	281	0.224627	83
parallel	0.008000	0.006005	14	9	756	0.083663	83
serial	0.003000	0.002247	15	11	338	0.187133	77
parallel	0.003000	0.002533	16	10	960	0.065787	77
serial	0.001350	0.001242	16	12	376	0.167766	73
parallel	0.001350	0.001315	17	11	1122	0.056173	73
serial	0.000350	0.000336	18	14	458	0.137642	65
parallel	0.000350	0.000339	19	13	1482	0.042531	65
serial	0.000100	0.000094	20	16	548	0.115073	57
parallel	0.000100	0.000089	21	15	1890	0.033392	57

Les résultats du Tableau 3.1 montrent que pour différentes architectures, notre méthode d'optimisation trouve des solutions différentes. Par exemple, une implantation sérielle de l'algorithme CORDIC, nécessite une unité opérative de 15 bits et 11 itérations afin de rencontrer une erreur maximale de 0.003, tandis que pour le même critère d'exactitude, une implantation parallèle est plus efficace avec une unité opérative de 16 bits exécutant 10 itérations.

Dans ce cas particulier, le fait d'implanter l'architecture parallèle avec 16 bits et 10 itérations plutôt qu'avec 15 bits et 11 itérations, permet de réduire le coût d'implantation matérielle H de 990[1] à 960. Cette situation est aussi observée pour l'architecture sérielle. Le fait d'implanter l'architecture sérielle avec 15 bits et 11 itérations plutôt qu'avec 16 bits et 10 itérations réduit son coût d'implantation matérielle H de 344[2] à 338. Pour tous les critères de précision présentés au Tableau 3.1, cette situation est observée.

[1] $H = 990$ est obtenu lorsque $N = 11$ itérations et $WL = 15$ bits sont reportés à l'équation 3.2
[2] $H = 344$ est obtenu lorsque $N = 10$ itérations et $WL = 16$ bits sont reportés à l'équation 3.1

Tableau 3.2 Solutions obtenues par la méthode de détermination automatique

Architecture	S_0	ε_0	a	x	y	Δa	Δx	Δy	N	H	C	Nb Sim.
serial	0.040000	0.039707	10	11	10	8	10	11	6	168	1.327425	1067
parallel	0.040000	0.039707	10	11	10	8	10	11	6	372	0.599482	1067
serial	0.025000	0.023764	10	12	10	8	10	12	8	188	1.186433	1039
parallel	0.025000	0.024670	12	10	11	10	11	10	7	462	0.482713	1032
serial	0.008000	0.007886	13	13	12	11	12	12	9	245	0.910262	955
parallel	0.008000	0.007886	13	13	12	11	12	12	9	684	0.326044	955
serial	0.003000	0.002957	15	15	15	13	15	15	10	303	0.736021	843
parallel	0.003000	0.002935	16	14	15	14	15	14	10	900	0.247802	843
serial	0.001350	0.001318	15	17	15	13	15	17	13	350	0.637211	794
parallel	0.001350	0.001322	17	16	16	15	16	16	11	1078	0.206884	780
serial	0.000350	0.000347	19	18	18	17	18	18	13	430	0.518625	682
parallel	0.000350	0.000347	19	18	18	17	18	18	13	1430	0.155950	682
serial	0.000100	0.000091	20	21	20	18	20	21	15	502	0.444402	584
parallel	0.000100	0.000099	21	20	20	19	20	20	15	1830	0.121863	584

Selon le coût d'implantation H qui modélise l'architecture, la méthode propose différentes alternatives afin de rencontrer les critères de précision, ce que ne propose pas Kota dans [37]. Les modèles analytiques, tel que celui-ci, sont pratiques pour les concepteurs matériels et permettent de donner une idée générale de ce dont auraient l'air les implantations matérielles. Cependant, ils restreignent les concepteurs à implanter les algorithmes avec un certain nombre de bits et d'itérations afin de rencontrer un critère de précision.

Pour la plupart des algorithmes itératifs, il n'existe pas de modèle analytique disponible dans la littérature, ce qui démontre l'importance de la méthode. Pour les algorithmes où il existe des modèles analytiques, ces modèles restreignent les chemins de données à des tailles spécifiques, ce qui limite l'espace de solution à des solutions qui exigent de plus grands coûts d'implantation. Cette autre observation est aussi présentée dans [12]. Le Tableau 3.2 montre les résultats obtenus par la méthode, pour les mêmes critères de précision que ceux montrés au Tableau 3.1, et lorsque les tailles des chemins de données WL_a, WL_x, WL_y, WL_{Δ_a}, WL_{Δ_x} et WL_{Δ_y} ne sont pas resteintent à la même valeur.

En comparant les Tableaux 3.1 et 3.2, il est possible d'observer que toutes les solutions trouvées au Tableau 3.2 requierent un coût d'implantation H moins élevé que celles du Tableau 3.1. Plus spécifiquement, le coût d'implantation a été réduit en moyenne de 9.5% pour l'architecture sérielle, et de 5.8% pour l'architecture parallèle.

Le fait d'accroître le nombre d'opérandes analysés I, permet à la méthode de trouver des solutions qui réduisent le coût matériel ou la consommation de puissance dans l'implantation finale de l'algorithme.

Le reste de ce chapitre présente (1) l'article intitulé "Automatic Word Length and Iteration Determination Method for the Hardware Optimization of Iteratives Algorithms", qui a été soumis au mois de mars 2005 à "IEEE Transaction on Circuits and Systems" et (2) une seconde modification apportée à la méthode.

3.2 Article: Automatic Word Length and Number of Iterations Determination Method for the Hardware Optimization of Iterative Algorithms

M.-A. Cantin, Y. Savaria, Y. Blaquière and N. Bélanger

Abstract

A method is proposed to determine, in a single optimization process, the combination of word lengths in hardware datapaths and the number of times that an iteration should be executed in iterative algorithms. The method searches for a combination of word length and number of iterations that minimizes the implementation cost and meets accuracy criteria specified by a hardware designer. The method was succesfully applied to several iterative algorithms and experimental results obtained for the CORDIC algorithm are presented. Using this example, the method finds, for different hardware architectures, different solutions that meet some specified accuracy objectives. This shows that for the same algorithm, the best solution may differ when the architecture is changed. Furthermore, by analyzing a larger number of hardware datapaths than with a method based on an analytic model found in the literature, it is shown that the method reduces the implementation cost by an average of 9.5%. Solutions were found without the need of tedious analytic developments, which is a significant advantage of the method, as for most iterative algorithms, no analytic model is available in the literature, and even when they could be found, they often restrict the hardware datapaths to some given word length and they prevent finding a solution with lower implementation costs.

3.2.1 Introduction

Optimizations made on a Digital Signal Processing (DSP) algorithm before implementing it can reduce power consumption [1], design time [2],[3], and hardware costs [4]. It can also lead to pure performance improvements [5]. One class of optimization is to determine the minimum number of times that an iterative algorithm should be executed.

Iterative algorithms such as the CORDIC algorithm [6], the Crozier frequency estimator [7], the iterative threshold decoder [8], and the Newton algorithm [9], execute iteratively a sequence of operations to yield outputs successively closer to the desired values. This iterative execution is repeated either for a specified number of times, or until a stopping criterion is met. Typically, in the first iteration, an estimation of the outputs is performed and, as long as the iterations proceed, the accuracy of the output results keeps improving. However, for a design implemented with a specific bit resolution, increasing the number of iterations does not necessarily increase the accuracy of the output results. When the possible accuracy supplied by an n^{th} iteration is greater than the bit resolution afforded by a word length combination, additional iterations will not provide further accuracy in the output results, and they can even decrease it. In this case, the accuracy of the output results can be increased only if the bit resolution of such hardware datapaths are increased.

Thus, for a given word length combination of hardware datapaths, an optimal number of iterations exists, and for a given number of iterations, an optimal word length combination exists. If the number of iterations in a design is larger than the optimal one, then the implementation will require a latency larger than necessary to produce output results, or the hardware implementation cost will be needlessly larger (as shown qualitatively in Fig.3.4). On the other hand, if the number of

iterations in a design is lower than the optimal one, then the implementation will usually produce less accurate output results.

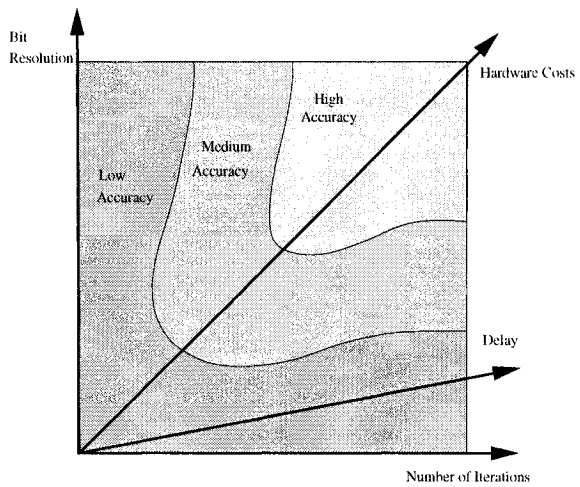


Figure 3.4 Several aspects of the implementation of iterative DSP algorithms

It can be beneficial to optimize the word lengths and the number of iterations in the same process to provide implementations reaching the specified accuracy with low hardware costs and small latency.

In this paper, an Automatic Word Length Determination (AWLD) method applicable to hardware datapaths is described. It determines, in the same process, both the bit resolution of the operands analyzed in iterative algorithms, and the number of iterations that should be executed. In the next section, previous work on word length determination is reviewed. In Section 3.2.3, our word length and number of iterations determination method is described and its application to the CORDIC algorithm is detailed in Section 3.2.4. Results produced by applying the proposed method to sequential and pipelined implementations of the CORDIC algorithm are provided and compared to an analytic model presented in [10]. Finally, our main conclusions are summarized in Section 3.2.6.

3.2.2 Word Length Determination of Hardware Datapaths

Digital Signal Processing (DSP) algorithms are usually coded in 32- or 64-bit fixed- or floating-point data formats, since they are available in most commercial off-the-shelf processors. Yet, in spite of very significant improvements in processor performance and power efficiency, the requirements of many real-time applications in terms of pure performance or low power operation still justify the use of specialized hardware. Cost, power dissipation, and performance of hardware implementations are highly dependent on the number of bits used to represent data, as well as on the use of floating-point operators. Word length determination is therefore a fundamental step in the design process. In some complex designs, half of the design time can be spent determining word lengths [11]. Moreover, many algorithms require a careful selection of word lengths to preserve their stability [12]. Powerful automatic and precise word-length determination methods are therefore required.

In the last 10 years, several techniques have been proposed to translate floating-point formulations into fixed-point formulations, especially for specific DSP applications [12],[13],[14]. Heuristic techniques and analytical methods dedicated to specific applications have been employed, but they cannot be generalized to other DSP applications.

For general DSP applications, the translation of floating-point formulations into fixed-point formulations consists of evaluating the dynamic range and the minimum accuracy of each operand O_i, by determining its Integer Word Length (IWL_i) and its Fractional Word Length (FWL_i), where $i = 0, 1, \ldots, (I-1)$, and where I is the number of operands to be translated. The resulting Word Length (WL) of each translated fixed-point operand O_i may then be obtained as follows:

$$WL_i = IWL_i + FWL_i + s_i \tag{3.3}$$

where $s_i = 0$ for unsigned and $s_i = 1$ for signed representations of O_i (two's complement is assumed in this paper). In Equation (3.3), the IWL and FWL can either be positive or negative. For example, a 4-bit binary data 1010 with $WL_i = 4$, $IWL_i = -2$, $FWL_i = 6$, and $s_i = 0$ should be interpreted as 0.001010, which corresponds to a decimal value of 0.15625. The same 4-bit binary data with $WL_i = 4$, $IWL_i = 6$, $FWL_i = -2$, and $s_i = 0$ will be interpreted as 101000, which corresponds to a decimal value of 40. Notice that $IWL + FWL \geq 0$ to ensure that the WL is greater than, or equal to 0.

3.2.2.1 Determination of the Integer Word Length (IWL)

Three known methods can be used to estimate the required IWL. The first method computes the IWL_i of each operand O_i as follows:

$$IWL_i = \lceil \log_2 (|O_i|^{\max}) \rceil \tag{3.4}$$

where $\lceil \cdot \rceil$ is the "ceiling" function and $|O_i|^{\max}$ is the maximum absolute value of the operand O_i extracted during simulations of the DSP algorithm [15],[16]. The second method computes each IWL_i like the first method, except that the maximum absolute value of the operand O_i is replaced by the values of O_i obtained by propagation of the maximum and minimum input values through a Data Flow Graph (DFG) of the DSP algorithm [17]. The third method, introduced by Sung et al. [18], extracts the mean μ_i and the standard deviation σ_i of the operand O_i during a simulation of the DSP algorithm, and then computes the IWL_i as follows:

$$IWL_i = \lceil \log_2 (\max (|\mu_i| + k \times \sigma_i, |O_i|^{\max})) \rceil \tag{3.5}$$

where k is an arbitrary constant and it is set equal to 4 in [18].

The first method does not guarantee that overflow will not occur in the data path, since the maximum value of some operands may not be stimulated during simulations. On the other hand, the second method is very fast, but it is also known to be a conservative approach that tends to overestimates IWL_i [19], even though it can also underestimate it if partial results cancel out. Furthermore, the DFG analysis requires a fixed-point specification of the input signals, and it becomes complex to manage when the algorithm contains data dependencies. Finally, the third method minimizes the probability of overflow in the data path when k is set to a high value.

3.2.2.2 Determination of the Fraction Word Length (FWL)

Three basic methods to determine the FWL are found in the literature. The first method evaluates the FWL by Data Flow Graph (DFG) analysis. FRIDGE [11] and CoCentric [20] propagate the FWLs through the DFG, while ensuring that no information is lost. This technique is called interpolation, and may be illustrated by a simple example: for $a = b + c$, no information is lost if $FWL(a) = \max(FWL(b), FWL(c))$. Alternatively, Yasuura *et al.*, in [16] and [17], perform numerical analysis on the DFG of the DSP algorithm. Determining the FWL using a DFG is fast, but it is a conservative approach that overestimates FWL [19]. Furthermore, as for the determination of the IWL, the DFG analysis requires a fixed-point specification of the input signals, and it becomes complex to manage when the algorithm contains data dependencies.

Cmar *et al.* [19] proposed a method that combines analytical rules and simulations. The method determines FWL_i based on measurements of the mean μ_i and standard deviation σ_i of the dissimilarities between fixed-point and floating-point simulations. Since the C language description of the DSP algorithm requires major

modifications for this method, the bit resolution analysis is performed only on one operand at a time, instead of on combinations of operands.

Finally, the FWL can be determined by pure simulation. This method computes the dissimilarities between the floating- and fixed-point simulation results, and based on them, a procedure searches for a combination of word lengths that meets some accuracy criteria specified by the user. This simulation-based method is known to be the most accurate, but in the past, this method was prohibitively time consuming for most applications. With recent improvements in processor performance, this method is now attractive for DSP algorithms with a small to medium complexity (in terms of the number of operands I).

Using this simulation based method, Sung & Kum [21], Han *et al.* [22], Choi and Burleson [23], Kirkpatrick *et al.* [24] and Fiore and Lee [25] have proposed manual procedures and guidelines to optimize the FWL. However, none of them offers fully automatic capability to determine in a single process the optimal word lengths of hardware datapath for several error models, user criteria and implementation costs. Therefore, a new automatic word length determination method was proposed by some of the authors in [26]. This method was implemented and applied to a dozen DSP algorithms in [27]. Furthermore, the manual procedures and guidelines found in the literature were adapted to our automated framework and compared for their ability to find good solutions rapidly [27].

None of the methods found in the literature considers optimizing the number of iterations needed by an iterative algorithm to reach a desired accuracy with a cost effective hardware implementation. The next section describes how to perform effective searches over parameter sets such as, the number of iterations to be performed, and the word size of operand needed to get a desired processing accuracy while minimizing implementation cost.

3.2.3 The Automatic Word Length and Number of Iterations Determination Method

An automatic word length determination method, that uses a simulation-based approach was developed by the authors. This method is identified as AWLD in the rest of this paper. The proposed method is built upon a fixed-point simulation tool and has been successfully implemented over two such tools: SystemC [28] and a fixed-point utility developed by W. Sung et al. [18]. These fixed-point simulation tools convert a floating-point DSP program written in C or C++ into a fixed-point equivalent description. Fixed-point arithmetic operations, rather than floating-point arithmetic, are conducted automatically using the operator overloading capability of C++. No modification of the original C/C++ program is needed by the AWLD method except for declaring which operands belong to a fixed-point class type, renaming the function *main* () and adding a fixed-point header file.

3.2.3.1 The AWLD Method

The AWLD method involves eight steps, as shown in Fig. 3.5.

1. Test bench generation

   The user generates test benches representative of the application. They must stimulate all operands, $\{O_i\}$, in the DSP algorithm, and all data paths of interest. The method considers I user-specified operands, and outputs a word length for each one. Each test bench is a file containing inputs used to stimulate the DSP algorithm.

2. Floating-point simulation

   A floating-point simulation of the DSP algorithm is performed for each test

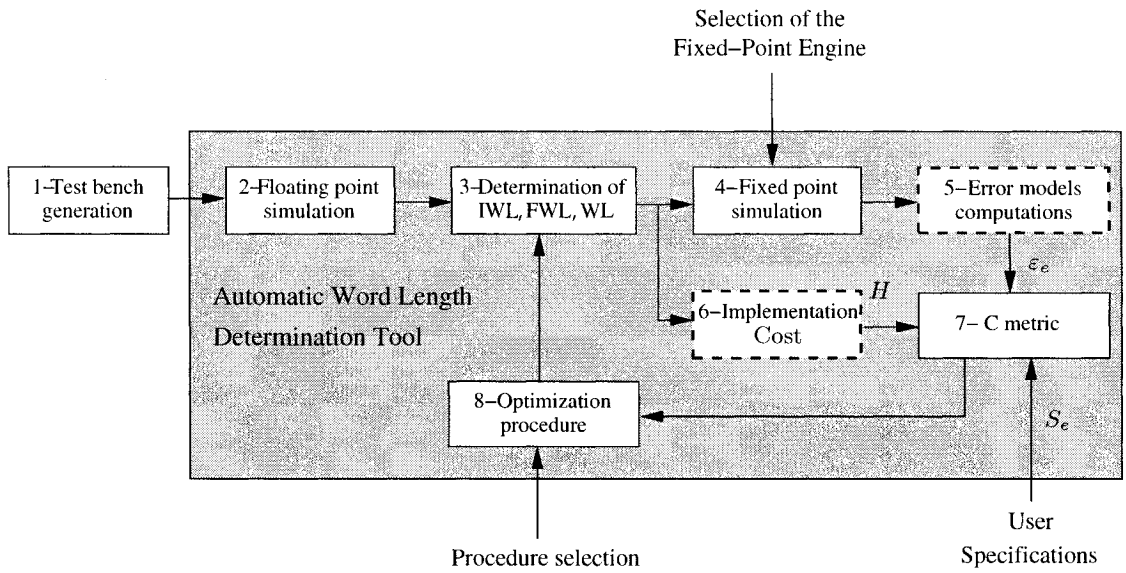


Figure 3.5 Automatic Word Length Determination Method of Hardware Datapaths

bench. Several parameters for each operand O_i are extracted and computed from the simulation: the maximum absolute value $|O_i|^{\max}$, the sign flag, s_i (see Equation (3.3)), the mean μ_i, and the variance σ_i. These parameters and output results produced by the DSP algorithm when processed in floating-point resolution are recorded for future reference.

3. Determination of the fixed-point operand formats

   For each operand O_i, the sign flag s_i is set equal to 0 if the operand O_i is always positive, or to 1 otherwise, and the integer word length IWL_i is set equal to the value defined in Equation (3.5)(page 105). The Fractional Word Length FWL_i is initialized according to the selected maximization procedure to be run in step 8 below.

4. Fixed-point simulation

   A fixed-point simulation is performed for each test bench. The output results produced by the DSP algorithm in fixed-point resolution are recorded.

5. Error model computations

Floating- and fixed-point simulation output results recorded in steps 2 and 4 respectively are used to compute the dissimilarity between the floating- and fixed-point representations according to various error computation models ε_e, for $e = 0, 1, \ldots, E-1$, where E is the number of considered error models. Error computation models ε_e, such as the maximum absolute error and mean square error between each correspondent floating- and fixed-point outputs, are provided to quantify the dissimilarities between the floating-point and fixed-point representations of the algorithm. However, the user is also allowed to provide his own error models ε_e by using predetermined C functions to get the floating- and fixed-point outputs results and to return the resulting ε_e values. To underline that the user is allowed to specify any function in the method, steps 5 and 6 are represented with dashed boxes in Fig. 3.5.

6. Implementation cost

   The implementation cost, denoted by H, is some measure of the hardware cost, power dissipation, or processing time for the word length combination $\{WL_i\}$, and is computed in the range $[1, \infty[$. By default, the implementation cost H is defined as follows:

$$H = \sum_{i=0}^{I-1} WL_i \tag{3.6}$$

   where WL_i is the total word length in bits of the i^{th} operand. Since Equation (3.6) is a simplification of reality and, to provide generality, the user is allowed to specify any suitable hardware cost function H based on the word length WL_i of each operand O_i.

7. Computation of the quality metric $\mathcal{Q}$

   Based on the error computations ε_e produced in step 5, the implementation

cost H produced in step 6, the user specifications S_e, a pass/fail threshold for each error computation ε_e, and d_e a pass/fail decision for each error computation, the following quality metric $\mathcal{Q}$ is computed.

$$\mathcal{Q} = \left[\sum_{e=0}^{E-1}\left(\frac{S_e - \varepsilon_e}{S_e}\right) d_e\right] \cdot H + \left[(I \cdot WL^{\max}) - \frac{1}{E}\sum_{e=0}^{E-1}\left(1 - \frac{S_e - \varepsilon_e}{S_e}\right)\right] \cdot \frac{1}{H}\prod_{e=0}^{E-1}(1 - d_e) \tag{3.7}$$

where $WL^{\max}$ is the maximum word length allowed by the fixed-point simulation tool, $d_e = 0$ passes if $S_e \geq \varepsilon_e$, and $d_e = 1$ fails if $S_e < \varepsilon_e$. Note that Equation (3.7) is composed by the addition of two terms that are mutually exclusive because of how d_e is used (as described below). The expression $\sum_{e=0}^{E-1}\left(\frac{S_e - \varepsilon_e}{S_e}\right)$ found in both terms grades all word length combinations and guides the search procedures toward the optimal solution in all cases. Furthermore, contributions of all user specifications are normalized by the metric. The first term of Equation (3.7), $\left[\sum_{e=0}^{E-1}\left(\frac{S_e - \varepsilon_e}{S_e}\right) d_e\right] \cdot H$, can only be null or negative. The metric is negative when at least one user specification is not met by the word length combination ($S_e < \varepsilon_e$). When one or more user specifications S_e are not met, $d_e = 1$, and they contribute to the first term. Otherwise, specifications that are met have no effect on the first term as their $d_e = 0$.

The second term of the metric is positive when all user specifications are met. Otherwise, one of the $\{d_e\}$ is equal to 1, in which case the right term is null. When all user specifications S_e are met, the main objective of the metric is reducing the implementation cost; this is achieved through the division of the second term by H. Furthermore, for two word length combinations that have the same implementation cost, the one that provides the lowest error has the highest $\mathcal{Q}$ value. This behavior comes from the term $(I \cdot WL^{\max}) -$

$\frac{1}{E}\sum_{e=0}^{E-1}\left(1-\frac{S_e-\varepsilon_e}{S_e}\right)$, where $(I \cdot WL^{\max})$ is a constant that ensures that the value of $\mathcal{Q}$ is larger than 0, and where the term $\sum_{e=0}^{E-1}\left(1-\frac{S_e-\varepsilon_e}{S_e}\right)$ decreases for word length combinations that reduce the error.

8. Maximization procedure

A maximization procedure tries to find the combination of word lengths $\{WL_i\}$ that maximizes the quality metric $\mathcal{Q}$ as defined in Equation (3.7). Nine procedures proposed in the literature were implemented in the AWLD method and were characterized in a comparison performed on a dozen of DSP algorithms [27]. From these procedures, the *Heuristic* [21] and *Hybrid* [27] procedures provided rapid and very good solutions most of the time. These two procedures are briefly described below.

The *Heuristic* procedure [21] proceeds in four phases. In the first phase, the minimum bit resolution that meets user specifications S_e is found for each operand O_i, $i = 0, 1, \ldots, (I-1)$, when all other operands O_j, $j = 0, 1, \ldots, (I-1)$, $j \neq i$, are in floating-point format. In the second phase of this procedure, the resolution of each operand is set equal to the value found in the first phase. This combination of word lengths is called the Minimum Word Length (MWL) combination (see Fig. 3.6). In the third phase, the *Heuristic* procedure successively increases all word lengths by one bit from the MWL combination until user specifications are met. In the fourth and last phase, the operand that generates the largest cost savings is selected to loose a bit for as long as the error specifications are still met.

The *Hybrid* procedure, proposed by the authors in [27], also comprises four phases. The first and second phase of the *Hybrid* procedure are the same as for the *Heuristic* procedure. In the third phase, an iterative competition takes place from the MWL combination between the operands to gain B bits

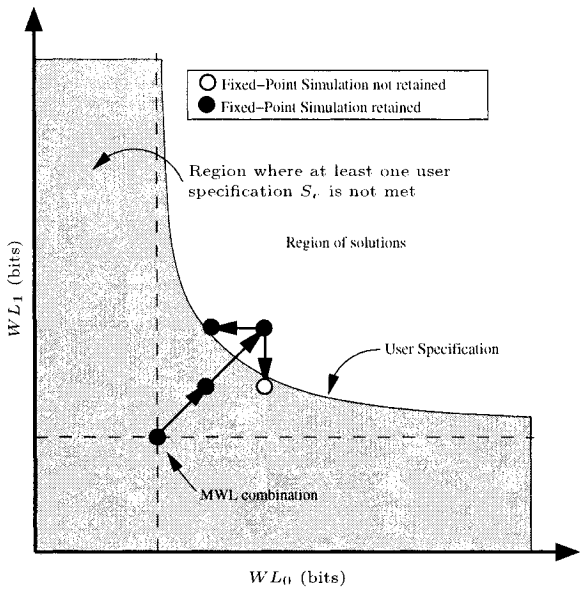


Figure 3.6 *Heuristic* procedure

(see Fig. 3.7). B bits are used instead of 1 bit because, for some DSP algorithms, it was found that increasing the bit resolution of a single operand at a time by 1 bit may fail to converge to an acceptable solution. This situation is explained below the description of the eigth steps of the AWLD method. The *Hybrid* procedure explores all L possible ways of assigning B bits to I operands, where L corresponds to the Pascal triangle, that is,

$$L = \frac{(I+B-1)!}{(I-1)!\,(B)!} \tag{3.8}$$

The value of B is initialized at 1, and is increased to $2, 3, \ldots$ only when needed (i.e. when the output accuracy, which is mesured by the error models ε_e, does not improve by adding 1 bit at a time to each operand), to avoid being trapped in a local maximum. Once the error is reduced, B is reset to 1, and the iterative competition is repeated until the user specifications are met. In the fourth and last phase of the *Hybrid* procedure, the operands compete to loose their bits in a second iterative competition. The quality

metric $\mathcal{Q}$ is computed with one bit removed from operand O_i, while all other operands O_j, $j \neq i$, remain unchanged. The operand O_i for which the quality metric $\mathcal{Q}$ is maximized wins the competition and looses its bit. This process is repeated iteratively as long as the error specifications are met.

The final solution is found when removing 1 bit from any single operand at a time fails to meet the error specification. Note however that, for some DSP algorithm, if two or more operands are simultaneously shortened, a better solution may be found. This happens when the quantization errors of more than 2 operands compensate for each other. Therefore, when removing 1 bit fails to yield a better solution, it may be worthwhile trying to remove $2, 3, \ldots, B$ bits simultaneously over the I operands. However, it is not clear what limit should be used for B in this case, thus in this last phase, B is restricted to 1 and this issue was left for future research. The last phase, which ensures that all word lengths are at their minimum value, allows the *Hybrid* procedure to reach solutions of lower implementation cost than the MWL combination.

As mentionned previously, increasing the bit resolution of a single operand at a time by 1 bit may fail to converge to an acceptable solution. This happens, for instance, when the overall accuracy increases only if two or more operands are simultaneously widened. For example, during the word length determination process of a two input adder $a = b + c$, let us assume that at some point the fraction word lengths of the operands are equal; $FWL_a = FWL_b = FWL_c$. If this word length combination does not meet the user specifications, then some word lengths must be increased. The maximum error of this adder was found empirically to be as given by Equation (3.9). The only way to reduce the maximum error is to widen simultaneously more than one operand at a time, because either (i) the decrease in the sum of the first two terms is compensated by an increase of the last term,

or (ii) when FWL_a is increased, the value of the max $(\cdot)$ function does not change.

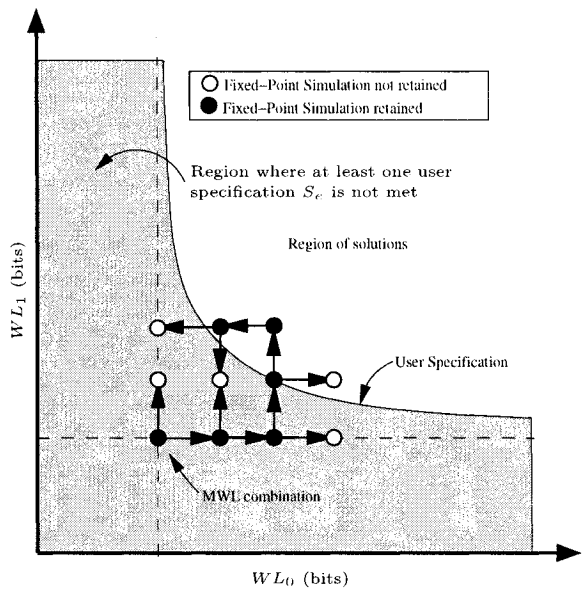


Figure 3.7 *Hybrid* procedure

$$\varepsilon_0 = 2^{-FWL_b} + 2^{-FWL_c} + \max\left(0,\ 2^{-FWL_a} - \min\left(2^{-FWL_b}, 2^{-FWL_c}\right)\right) \qquad (3.9)$$

For example, if $FWL_a = FWL_b = FWL_c = 3$, then the maximum truncation error ε_0 that is observed at the output is equal to 0.250. Adding one bit to one of these operands leaves ε_0 unchanged to $\varepsilon_0 = 0.250$, because (i) the reduction of the error produced by one of the inputs (FWL_b or FWL_c) cannot be propagated to the output (FWL_a), or (ii) the output (FWL_a) affords one additional bit that can never be used by the inputs (FWL_b or FWL_c). Thus, to minimize ε_0, one bit to FWL_a and FWL_b, FWL_a and FWL_c, or FWL_b and FWL_c will be required to produce $\varepsilon_0 = 0.1875$. Therefore, when adding 1 bit fails to provide significant accuracy improvements, $2, 3, \ldots, B$ bits are temporarily distributed on the I operands until the error decreases.

3.2.3.2 Adapting the AWLD method to determine the number of iterations

The number of iterations in an iterative algorithm has a similar impact on the overall accuracy as the word length of an operand O_i, i.e. when they are increased, the overall accuracy of the algorithm tends to increase and when they are decreased, the accuracy tends to decrease. Based on this behavior, the automatic word length determination method was modified in order to determine, in a single process, the word lengths and the number of iterations. The number of iterations in an iterative algorithm can be assimilated to the word length of a new operand O_i with the following modifications in the formulation of the AWLD method presented in Section 3.2.3.1:

1. The declaration of the variables that determine the number of iterations belongs to another C++ class;
2. In the floating-point simulation, variables that determine the number of iterations are set to $WL_i^{\max}$, the maximum word length allowed by the fixed-point tool;
3. The proposed method sets the value of variables that determine the number of iterations at their declaration and user assignment to these variables during the execution of the DSP algorithm is prohibited.

This new automatic method was implemented and successfully applied to iterative DSP algorithms such as the CORDIC [6] algorithm, the Crozier frequency estimator [7], the iterative threshold decoder [8], and the Newton algorithm [9]. Since the CORDIC algorithm is a well known algorithm, and since results obtained by the method for this algorithm are representatives of those obtained by the previously cited iterative algorithms, this paper presents the results for the CORDIC

algorithm. Note that, in the CORDIC algorithm, there is only one number of iterations to determine. However, if an algorithm contains severals iterative loops, the method was also built in order to determine them in a single process.

As this paper focuses on the CORDIC algorithm used as a benchmark, it is of interest to review this algorithm. The CORDIC algorithm, some existing hardware architectures implementing it (one serial and one parallel), and the various data paths that compose them, are briefly reviewed in Section 3.2.4.

3.2.4 The COordinate Rotation Digital Computing (CORDIC)

The COordinate Rotation Digital Computing (CORDIC) algorithm, which was first developed by Volder [6], is a well-known algorithm in the DSP domain. It allows to compute some complex trigonometric functions such as the sine, the cosine, the inverse tangent and the square root, with simple hardware. It is based on the following recurrent equations:

$$x_{n+1} = x_n - d_n \cdot \Delta y_n \tag{3.10}$$

$$y_{n+1} = y_n + d_n \cdot \Delta x_n \tag{3.11}$$

$$a_{n+1} = a_n - d_n \cdot \Delta a_n \tag{3.12}$$

where

$$\Delta x_n = x_n \cdot 2^{-n} \tag{3.13}$$

$$\Delta y_n = y_n \cdot 2^{-n} \tag{3.14}$$

$$\Delta a_n = \tan^{-1} 2^{-n} \tag{3.15}$$

$\Delta x_n, \Delta y_n$, and Δa_n are defined (as opposed to merging equations 3.13 to 3.15 with equations 3.10 to 3.12) because, in the AWLD process, they could require lengths different from x, y, and a respectively. Note that the Δa_n values can be precomputed and stored in a ROM.

The CORDIC algorithm can be implemented using a serial or a parallel architecture. A simple serial architecture proposed in [29] is shown in Fig. 3.8. It includes, for the implementation of the rotation mode only, 3 adder-subtractors, 3 registers, 2 shifters, 3 multiplexers, 1 counter of $\log_2 N$ bits, and one look up table to store the N precomputed values Δa_i. The serial architecture can provide one result every N cycles. The parallel architecture [29], shown in Fig. 3.9, has a higher hardware cost than the serial one when the number of iterations, N, is high. It requires $(3 \cdot N)$ registers and adder/subtractors. As for the serial architecture, the parallel architecture has a latency of N cycles to produce the result of one angle, but produces one result and take a new angle at each clock cycle.

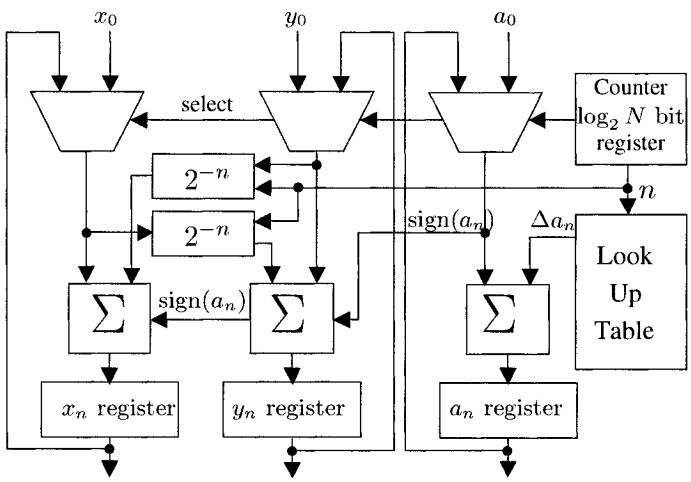


Figure 3.8 Serial implementation of the CORDIC algorithm

Table 3.3 summarizes the characteristics of the two architectures, where the throughput and the hardware costs are highly dependent on the architecture and the number of iterations.

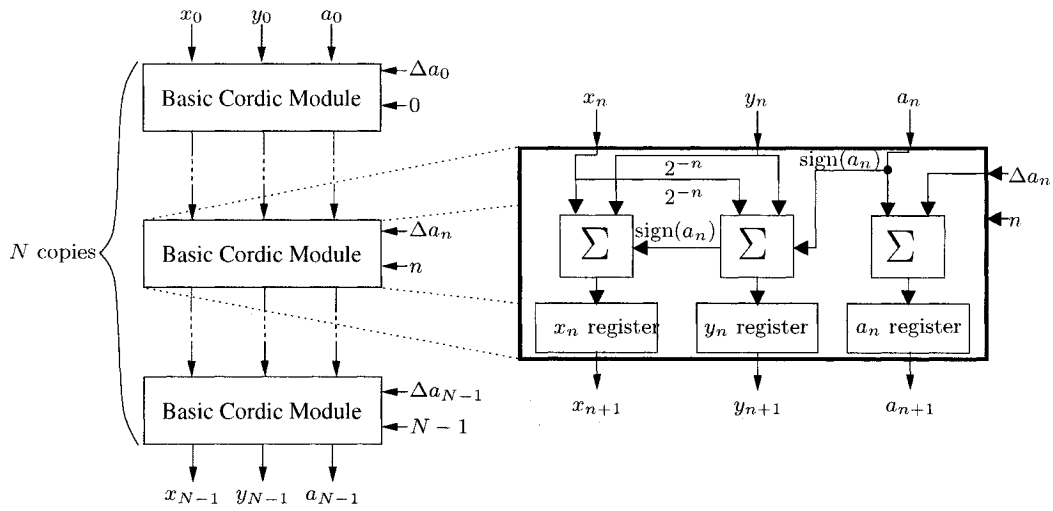


Figure 3.9 Parallel implementation of the CORDIC algorithm

Tableau 3.3 Summary of the CORDIC architectures

Architecture	Throughput	Latency	Hardware Costs
Serial	1 result/N cycles	N	3 adder/subtractors, 3 registers, 2 shifters, 3 multiplexers, 1 counter, 1 lookup table
Parallel	1 result/cycle	N	$(3 \cdot N)$ registers and adder/subtractors

3.2.5 Accuracy of the CORDIC algorithm

The CORDIC algorithm is a good example of an iterative algorithm where the accuracy is dictated by the number of elementary rotation, N, and the word length of the datapaths x, y, and a. Kota presents in [10] an analytic model of the CORDIC algorithm, which provides solutions that are compared with those provided by our method. Kota [10] concluded that the word length of datapaths x, y, and a, when restricted to the same lengths, and when the CORDIC algorithm operates in the range $[-\pi/2, \pi/2]$, need a CORDIC arithmetic unit of $N + \log_2 N + 2$ bits with N iterations to obtain N bits of accuracy in the results. As an example, Kota showed that 8 bits of accuracy at the outputs requires a CORDIC arithmetic unit with 11 bits of datapath width and 8 iterations [10].

However, results obtained by our method demonstrate that, unlike what is assumed

with the analytic model, when different architectures implementing the CORDIC algorithm are analyzed, different solutions are needed to meet some given resolution while minimizing hardware costs. In order to compare results produced by our method and the ones obtained by the analytic model presented in [10], the word lengths $WL_a, WL_x, WL_y, WL_{\Delta a}, WL_{\Delta x}$ and $WL_{\Delta y}$ have been restricted to the same size. Fig. 3.10 shows the contour plots of the maximum error produced by both the serial and parallel architectures of the CORDIC algorithm. The maximum error is reported when the number of iterations, N, varies from 5 to 18, and when all the word lengths contained in the CORDIC algorithm are set equal as assumed in [10] and vary from 8 to 22 bits. Note that only grid points correspond to feasible solutions. Fig. 3.10 was computed using Matlab to interpolate the feasible solutions and produce contour plots that are easier to visualize. These curves were obtained for a simulation of 1×10^6 random angles in the range $[-\pi/2, \pi/2]$. From these curves, it can be seen that, in general, the higher the number of iterations N and the word length of the CORDIC arithmetic unit are, the smaller the maximum error observed is. Conversely, the smaller the number of iterations and the word length of the CORDIC arithmetic unit are, the higher the maximum error is.

For a given number of iterations, increasing the word length of the arithmetic unit increases the accuracy of the output results (e.g. from point M to point P in Fig. 3.10). However, when the accuracy provided by an n^{th} bit is larger than the bit resolution afforded by the number of iterations, additional bits will not improve accuracy in the output results (e.g. from A to B). In this case, the accuracy of the output results can be increased only if the number of iterations is increased (e.g. from A to D). This observation is also true when the number of iterations changes for a given word length at the exception that near the minimum resolution afforded by a given word lengths, increasing the number of iterations could also reduce the accuracy, which is counterintuitive (e.g. from V to Y). For

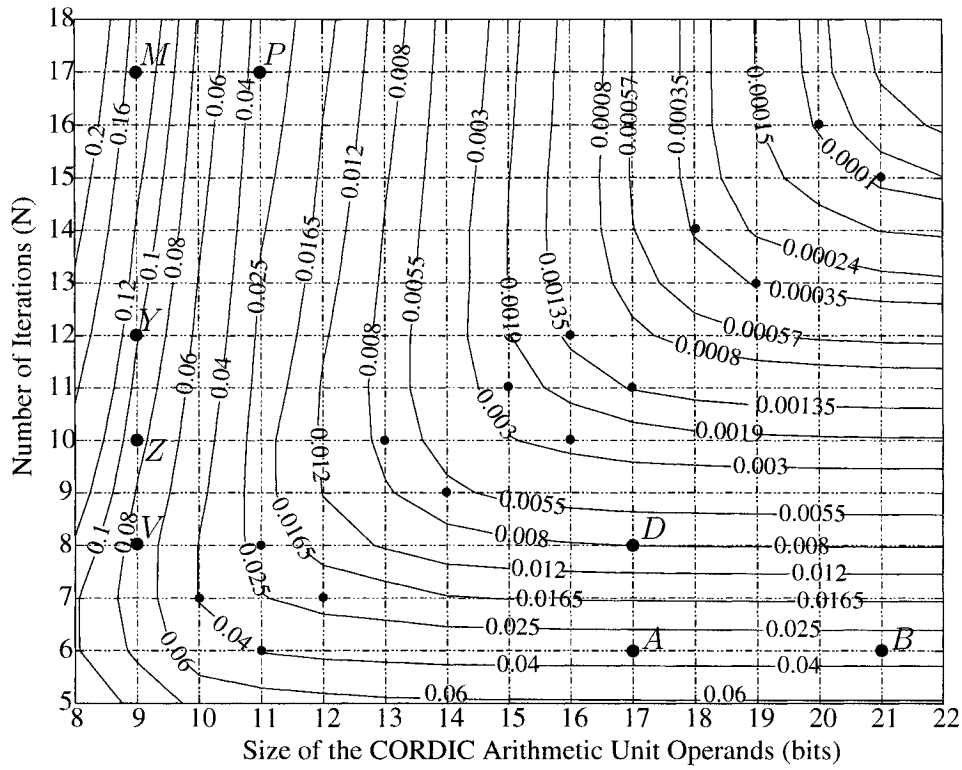


Figure 3.10 Maximum error of the CORDIC algorithm between the floating-point and fixed-point models

example, the maximum error observed in this simulation with an arithmetic unit of 9 bits and 8 iterations is equal to 0.070. If two more iterations are performed, then the maximum error is increased to 0.086 (from V to Z). Thus, the number of iterations should be carefully determined in relation to the word length of the datapaths. This counterintuitive result was also found several times for the word length determination of two or more operands. It is observed for instance when the quantization errors contributed by one operand or more compensate the error produced by the finite number of iterations.

Table 3.4 shows the results produced by our method for various accuracy specifications. It reports the word length of datapaths and the number of iterations for

Tableau 3.4 Implementation solutions of the CORDIC algorithm provided by the Automatic Word Length Determination Tool, when all hardware datapaths are restricted to the same size.

Architecture	Objective S_0	Error ε_0	WL_a, WL_x, and WL_y	N	H	Quality $\mathcal{Q}$	Nb sim.
serial	0.040000	0.039272	10	7	186	0.338808	95
parallel	0.040000	0.039069	11	6	396	0.159150	95
serial	0.025000	0.019854	11	8	215	0.293981	91
parallel	0.025000	0.020764	12	7	504	0.125336	91
serial	0.008000	0.007039	13	10	281	0.224627	83
parallel	0.008000	0.006005	14	9	756	0.083663	83
serial	0.003000	0.002247	15	11	338	0.187133	77
parallel	0.003000	0.002533	16	10	960	0.065787	77
serial	0.001350	0.001242	16	12	376	0.167766	73
parallel	0.001350	0.001315	17	11	1122	0.056173	73
serial	0.000350	0.000336	18	14	458	0.137642	65
parallel	0.000350	0.000339	19	13	1482	0.042531	65
serial	0.000100	0.000094	20	16	548	0.115073	57
parallel	0.000100	0.000089	21	15	1890	0.033392	57

the serial and parallel architectures. These results were obtained when the implementation cost, H, is set equal to Equation (3.16) for the serial architecture and to Equation (3.17) for the parallel architecture, with the simplifying assumption that the hardware costs of 1 bit of a register, 1 bit of an adder, 1 bit of a multiplexor, 1 bit of a counter, 1 bit of a shifter and 1 bit of a look up table are set equal to 1.

$$H = ((3 \cdot WL_x) + WL_{\Delta x} + (3 \cdot WL_y) + WL_{\Delta y} + (3 \cdot WL_a)) + (2 \cdot \lceil \log_2 (N) \rceil) + (WL_{\Delta a} \cdot N) \tag{3.16}$$

$$H = N \cdot ((2 \cdot WL_x) + (2 \cdot WL_y) + (2 \cdot WL_a)) \tag{3.17}$$

Table 3.4 shows that the method found, for some given objectives, different solu-

tions for the two hardware architectures. For instance, a serial CORDIC implementation with an arithmetic unit of 15 bits and 11 iterations, meets the maximum error objective of 0.003 when a parallel CORDIC implementation is more effectively implemented and meets also the maximum error objective of 0.003 with 16 bits operands and using 10 iterations. In this particular case, the fact that the parallel implementation can be implemented with 16 bit datapaths and 10 iterations instead of 15 bit datapaths and 11 iterations reduces the hardware cost from 990[3] to 960. This situation is also true for the serial architecture. The fact that the serial implementation can be implemented with 15 bit datapaths and 11 iterations instead of 16 bit datapaths and 10 iterations reduces the hardware cost from 344[4] to 338. For all the objectives presented in Table 3.4, these observations apply. Note that this difference in the results was expected, since the hardware cost of the parallel architecture is more affected by the number of iterations than the hardware cost of the serial architecture.

The automatic word length and number of iterations method can provide different alternatives to meet accuracy objectives according to the implementation cost H and the architecture model used to implement the iterative algorithm. Such alternatives are not provided by the analytic model presented in [10] for the CORDIC algorithm. This analytic model is useful for hardware designers and provides a guideline for hardware implementation, although it constrains the hardware designer to implement the CORDIC algorithm with a fixed number of iterations and arithmetic unit word length for a given accuracy at the output.

Global setting of word lengths of severals hardware data paths to the same value, as was assumed Kota [10] in his analytic model, may prevent finding solutions with

[3] $H = 990$ is obtained when $N = 11$ iterations and $WL = 15$ bits are reported in Equation (3.17)

[4] $H = 344$ is obtained when $N = 10$ iterations and $WL = 16$ bits are reported in Equation (3.16)

Tableau 3.5 Implementation solutions of the CORDIC algorithm provided by the Automatic Word Length Determination Tool, when all hardware datapaths are not restricted to the same size.

Architecture	S_0	ε_0	Word Length WL (bits)						N	H	$\mathcal{Q}$	Nb Sim.	ΔH (%)
			a	x	y	Δa	Δx	Δy					
serial	0.040000	0.039707	10	11	10	8	10	11	6	168	1.3274	1067	9.68
parallel	0.040000	0.039707	10	11	10	8	10	11	6	372	0.5994	1067	6.06
serial	0.025000	0.023764	10	12	10	8	10	12	8	188	1.1864	1039	12.56
parallel	0.025000	0.024670	12	10	11	10	11	10	7	462	0.4827	1032	8.33
serial	0.008000	0.007886	13	13	12	11	12	12	9	245	0.9102	955	12.81
parallel	0.008000	0.007886	13	13	12	11	12	12	9	684	0.3260	955	9.52
serial	0.003000	0.002957	15	15	15	13	15	15	10	303	0.7360	843	10.36
parallel	0.003000	0.002935	16	14	15	14	15	14	10	900	0.2478	843	6.67
serial	0.001350	0.001318	15	17	15	13	15	17	13	350	0.6372	794	6.91
parallel	0.001350	0.001322	17	16	16	15	16	16	11	1078	0.2068	780	3.92
serial	0.000350	0.000347	19	18	18	17	18	18	13	430	0.5186	682	6.11
parallel	0.000350	0.000347	19	8	18	17	18	18	13	1430	0.1559	682	3.51
serial	0.000100	0.000091	20	21	20	18	20	21	15	502	0.4444	584	8.39
parallel	0.000100	0.000099	21	20	20	19	20	20	15	1830	0.1218	584	3.17

lower implementation costs. For instance, this situation is observed when word lengths $WL_a, WL_x, WL_y, WL_{\Delta a}, WL_{\Delta x}$ and $WL_{\Delta y}$ of the CORDIC algorithm are not set equal. Table 3.5 shows results produced by our method for the two considered architectures and for the same accuracy objectives specified in Table 3.4.

By comparing results of Table 3.4 and Table 3.5, solutions reported in Table 3.5 required less hardware costs than the ones reported in Table 3.4. Specifically, from Table 3.4 to Table 3.5, the implementation cost H was reduced by an average of 9.5% and 5.8% for the serial and parallel architecture respectively (see column ΔH in Table 3.5), which is a relatively significant improvement for the two architectures. Thus, increasing the number of operands analyzed allows the method to find solutions that can reduce latency, hardware costs or power dissipation in the final implementation of the algorithm.

For each solution shown in Table 3.5, the method took an average of 20.7 hours and 850.5 fixed-point simulations to find a solution, where one fixed-point simulation is performed in 87.6 seconds on a 2.66GHz Intel P4 processor with 1GByte of memory

and running under the Linux operating system. Considering that 7 operands are analyzed and each operand can be defined with 1 to 32 bits, there are up to $32^7 = 3.44 \times 10^{10}$ different ways to implement one architecture of the CORDIC algorithm. Exaustive analysis of all possible configurations is then obviously impractical. The proposed automatic word length method found solutions that meet the accuracy specifications while minimizing hardware costs with a number of simulations much smaller than the number required for an exaustive coverage of the search space.

Finally, for most iterative algorithms, there are no analytic model available, which demontrates the importance of an automatic method that determines in a single optimization process the word length of hardware datapaths and the number of iterations that should be executed, for a desired output accuracy, hardware architecture and implementation costs.

3.2.6 Conclusion

A method has been implemented to determine automatically and simultaneously the word length of hardware datapaths and the number of times that a sequence of operations should be executed in iterative DSP algorithms. This automatic word length and number of iterations determination method is based on the fact that the number of iterations has a same behavior as the width of a hardware an operand in a hardware datapath. Indeed, as they are increased, they tend to improve the accuracy of the output results.

The automatic word length and number of iterations method was implemented, applied to the CORDIC algorithm and it succesfully provided optimized solutions for two different hardware architectures of the CORDIC algorithm. These solutions were found without the need of complex analytic models.

This paper has shown that the modified automatic word length determination

method is able to determine, in a single process, the word length of hardware datapaths and the number of times that an iteration should be executed for iterative algorithms, to meet an accuracy specification, while minimizing the implementation cost of the design.

References

[1] Wen-Tsong Shiue. Low power vlsi design: An optimal binding algorithm in high-level synthesis using integer linear programming. *Proceedings of the World Multiconference on Systemics, Cybernetics and Informatics*, vol. 15:458–461, 2002.

[2] M.B.E. Abdelrazik. A parallel multidimensional digital signal processing technique. *International Conference on Digital Processing of Signals in Communications*, pages 268–271, 1991.

[3] M.L. Chang and S. Hauck. Precis: a design-time precision analysis tool. *IEEE Symposium on Field-Programmable Custom Computing Machines*, pages 229–238, 2002.

[4] K. Okada, A. Yamada, and T. Kambe. Hardware algorithm optimization using Bach C. *IEICE Transactions on Fundamentals of Electronics, Communications and Computer Sciences*, vol.E85-A(4):835–841, 2002.

[5] Qiao Xiangzhen. Cache performance and algorithm optimization. *Proceedings of the High Performance Computing on the Information Superhighway HPC Asia*, pages 12–17, 1997.

[6] J.E. Volder. The CORDIC Trigonometric Computing Technique. *IRE Transaction on Electronic Computers*, vol.EC-8(3):330–334, 1959.

[7] S.N. Crozier. Performance and Complexity of Discrete-Time Frequency Estimation Algorithms. *17th Biennial Symposium on Communications, Queen's University, Ontario, Canada*, 1994.

[8] C. Cardinal, D. Haccoun, and F. Gagnon. Iterative threshold decoding without interleaving for convolutional self-doubly orthogonal codes. *IEEE Transactions on Communications*, vol.IT-51(8):1274–1282, 2003.

[9] C.F. Van Loan. *Introduction to Computational Science and Mathematics.* Jones and Bartlett Publishers, 1996.

[10] K. Kota and J. R. Cavallaro. Numerical Accuracy and Hardware Tradeoffs for CORDIC Arithmetic for Special-Purpose Processors. *IEEE Transactions on Computers*, vol. 42(7):769–779, 1993.

[11] H. Keding, M. Willems, M. Coors, and H. Meyr. FRIDGE: a fixed-point design and simulation environment. *Design, Automation and Test in Europe*, pages 429–435, 1998.

[12] M.-A. Cantin, Y. Blaquière, Y. Savaria, P. Lavoie, and E. Granger. Analysis of quantization effects in a digital hardware implementation of a fuzzy ART neural network algorithm. *IEEE Int. Symposium on Circuits and Systems*, vol. 3:141–144, 2000.

[13] J. Yli-Kaakinen and T. Saramaki. An efficient algorithm for the design of lattice wave digital filters with short coefficient wordlength. *International Symposium on Circuits and Systems*, vol. 3:443–448, 1999.

[14] I.-T. Lim and J. Bahn. Optimal wordlength determination of AC-3 decoding hardware based on fixed-point analysis and simulations of AC-3 algorithm. *IEEE Workshop on Signal Processing*, pages 301–310, 1997.

[15] T. Aamodt. Floating-Point to Fixed-Point Compilation and Embedded Architectural Support. Master's thesis, University of Toronto, 2001.

[16] H. Yamashita, H. Yasuura, F. N. Eko, and C. Yun. Variable Size Analysis and Validation of Computation Quality. *Proceedings of Workshop on High-Level Design Validation and Test*, pages 95–100, 2000.

[17] S.A. Wadekar. Accuracy Sensitive Word-length Selection for Algorithm Optimization. *Proceedings of the International Conference on Computer Design: VLSI in Computers and Processors*, pages 54–61, 1998.

[18] S. Kim, Ki-Il Kum, and W. Sung. Fixed-Point Optimization Utility for C and C++ Based Digital Signal Processing Programs. *IEEE Transaction on Circuits and Systems II*, vol. 45(11):1455–1464, 1998.

[19] R. Cmar, L. Rijnders, P. Schaumont, S. Vernalde, and I. Bolsens. A methodology and design environment for DSP ASIC fixed point refinement design. *Automation and Test in Europe Conference and Exhibition*, pages 271–276, 1999.

[20] Synopsys Inc. *Press Release: Synopsys Accelerates System-Level C-Based DSP Design With CoCentric Fixed-Point Designer Tool*, 2000.

[21] W. Sung and K. Kum. Simulation-based word-length optimization method for fixed-point digital signal processing systems. *IEEE Transaction on Signal Processing*, vol. 43(12):3087–3090, 1995.

[22] K. Han, I. Eo, K. Kim, and H. Cho. Numerical word-length optimization for CDMA demodulator. *IEEE International Symposium on Circuits and Systems*, vol. 4:290–293, 2001.

[23] H. Choi and W. P. Burleson. Search-based wordlength optimization for VLSI/DSP synthesis. *VLSI Signal Processing VII*, pages 198–207, 1994.

[24] S. Kirkpatrick, C.D. Gelatt, and M.P. Vecchi. Optimization by simulated annealing. *Science*, 220:671–680, 1983.

[25] P.D. Fiore and Li Lee. Closed-form and real-time wordlength adaptation. *Proceedings of the IEEE International Conference on Acoustics, Speech and Signal Processing*, vol. 4:1897–1900, 1999.

[26] M.-A. Cantin, Y. Savaria, D. Prodanos, and P. Lavoie. An automatic word length determination method. *IEEE International Symposium on Circuits and Systems*, vol. 5:53–56, 2001.

[27] M.-A. Cantin, Y. Savaria, and P. Lavoie. A Comparison of Automatic Word Length Optimization Procedures. *IEEE International Symposium on Circuits and Systems*, vol. 2:612–615, 2002.

[28] M. Speitel and B. Niemann. SystemC design language for development of ASICs and systems. *Elektronik*, vol. 50(13):78–83, 2001.

[29] G. A. Hampson. *Implementing Multi-Dimensional Digital Hardware Beamformers*. PhD thesis, Monash University, Department of Robotics and Digital Technology, 1997.

3.3 Optimisation de paramètres

La seconde adaptation apportée à la méthode a permis de déterminer simultanément, pour certaines applications spécifiques, les tailles des chemins de données et la valeur des paramètres contenus dans les algorithmes.

Sachant que les paramètres contenus dans les algorithmes de traitement de signaux sont souvent déterminés empiriquement [55][45][40], l'utilité d'une méthode qui estime automatiquement la valeur optimale de ces paramètres, devient importante. De plus, une méthode qui détermine simultanément la taille des chemins de données et les valeurs des paramètres, permettrait d'ajuster plus précisément la valeur de ces paramètres sous l'influence de la résolution virgule-fixe de l'algorithme.

Trois modifications majeures ont été apportées à la méthode afin de considérer l'optimisation des paramètres. La première modification a été apportée sur la structure même de la méthode. Vu que les résultats obtenus lors de l'étape de la simulation virgule-flottante sont considérés comme une référence pour les étapes subséquentes de la méthode, il faut donc s'assurer que cette simulation produit les meilleurs résultats possibles. C'est pourquoi, une étape additionnelle a été ajoutée avant la simulation virgule-flottante (voir Figure 1.1), lorsque des paramètres sont analysés par la méthode. Cette nouvelle étape consiste à effectuer une première optimisation de ces paramètres, pendant que la résolution de tous les autres opérandes (chemins de données) ont une résolution virgule-flottante.

La seconde modification a été apportée sur les procédures de recherche. Celles-ci augmentent la taille des chemins de données lorsqu'elles cherchent à augmenter la résolution des résultats, et elles diminuent la taille des chemins de données lorsqu'elles cherchent à diminuer la résolution des résultats. Cependant, cette technique ne peut pas être appliquée aux paramètres d'un algorithme. En fonction de

l'algorithme, de son domaine d'application et des valeurs initiales des paramètres, l'augmentation de la qualité des résultats peut aussi bien être obtenu en augmentant ou en diminuant les valeurs de ceux-ci. Par conséquent, à chaque fois que la valeur d'un paramètre est sujet à être modifié, toutes les procédures disponibles dans la méthode ont été modifiées afin de considérer simultanément l'augmentation et la diminution de ceux-ci.

La dernière modification a été apportée sur la déclaration des variables qui définissent les paramètres à optimiser dans l'algorithme. Cette déclaration doit comporter trois nouveaux arguments, dont la justification est donnée ci-après. Ne connaissant pas la plage dynamique de chaque paramètre, l'usager doit spécifier lors de la déclaration des paramètres analysés, les deux premiers arguments additionnels qui déterminent la valeur minimale $P_l^{\min}$ et la valeur maximale $P_l^{\max}$ que peuvent prendre ces paramètres, où $l = 0, 1, \ldots, L-1$, avec L le nombre de paramètres analysés. Dès lors, la procédure peut initialiser tous les paramètres à $\left(P_l^{\min} + P_l^{\max}\right)/2$. Par la suite, la procédure sélectionnée peut entreprendre l'optimisation des paramètres par l'ajout de plus ou moins S_l qui est initialisé à $S_l = \left(P_l^{\min} + P_l^{\max}\right)/WL^{\max}$, où $WL^{\max}$ est la résolution maximale permise par le simulation virgule-fixe. Lorsqu'une procédure tente d'optimiser la valeur d'un paramètre et que l'ajout de plus ou moins S_l n'améliore pas la qualité des résultats, S_l est réduit de moitié, $S_l = S_l/2$, et la tentative d'optimisation peut être reprise. Cette diminution de S_l s'effectue temps et aussi longtemps que S_l est supérieur à la valeur minimale $S_l^{\min}$ qui est spécifié par l'usager (le troisième argument).

Ces trois modifications ont permis de déterminer avec succès dans [52], les valeurs des paramètres d'un décodeur à seuils itératifs [16], et elles ont aussi permis de déterminer avec succès dans [21], les valeurs des paramètres et la taille des chemins de données d'un algorithme de dé-entrelacement d'images [41].

CONCLUSION GÉNÉRALE

Dans le cadre de cette thèse, une nouvelle méthode qui détermine automatiquement la taille des chemins de données a été présentée. Contrairement aux méthodes proposées dans la littérature, cette méthode permet d'optimiser automatiquement la taille des chemins de données en tenant compte simultanément de plusieurs mesures de précision, critères de précision et coûts d'implantation. Cette méthode est basée sur une métrique qui grade toutes les combinaisons des tailles des chemins de données vers la solution optimale. Une procédure qui optimise cette métrique minimise l'écart entre les modèles virgule flottante et virgule fixe et obtient une solution. Quatre nouvelles procédures de recherche ont été présentées: $Min+b\ bits$, $Max-1\ bit$, *Evolutive* et *Hybride*. Ces procédures ont été comparées à celles retrouvées dans la littérature qui ont été adaptées et implantées dans la méthode. La comparaison a permis de déterminer que les procédures *Hybride* et *Heuristique* sont celles qui sont les plus aptes à trouver rapidement des solutions optimisées. Avec ces procédures, la méthode permet de diminuer le temps de conception, le coût d'implantation, la consommation de puissance tout en maximisant les performances des algorithmes.

Il a été montré que les erreurs engendrées par les représentations en virgule fixe des opérandes peuvent s'accumuler ou se compenser. Lorsque les erreurs se compensent, une situation contre-intuitive est observée: l'ajout de bits supplémentaires à un opérande dans la partie fractionnaire FWL peut diminuer la précision des résultats.

La méthode a été appliquée avec succès sur différents types d'algorithme de traitement de signaux dont: le filtre elliptic du 5^e ordre [24], le réseau de neurones Fuzzy ART [17], l'algorithme CORDIC [60], l'estimateur fréquentiel de Crozier [26], un décodeur à seuils itératifs [16], la transformée discrète de cosinus [47] et son

inverse [47], l'algorithme ELA de dé-entrelacement des images [40], des filtres graphiques (SUSAN [55], Hybride [45], Weiner [43]) et un égalisateur linéaire transversal [36]. Plus en détails, une plate-forme publiée dans [9] a été présentée. Cette plate-forme utilise la méthode afin d'analyser les algorithmes de traitement vidéo. De plus, cette plate-forme permet de distribuer l'analyse sur plusieurs ordinateurs ce qui diminue considérablement le temps de calcul. Une autre technique élaborée dans [52] propose une alternative afin d'accélérer l'optimisation des tailles des chemins de données. Celle-ci exécute l'algorithme de traitement de signal directement sur son implantation matériel. Cette approche a permis d'effectuer une optimisation d'un décodeur à seuils itératifs en 1.58 heures, qui autrement aurait nécessité 321.16 jours de simulation sur un Sparc V440.

Finalement, la méthode a été adaptée afin de l'appliquer à deux autres types d'optimisation. Dans un premier temps, la méthode a été adaptée afin d'optimiser les algorithmes itératifs. Cette adaptation permet de déterminer dans un même processus, la taille des chemins de données et le nombre de fois qu'une itération doit être effectuée. La méthode adaptée a permis d'optimiser avec succès plusieurs algorithmes itératifs dont l'algorithme CORDIC [60], l'estimateur fréquentiel de Crozier [26], un décodeur à seuils itératifs [16] et l'algorithme de Newton [44]. Les solutions obtenues pour l'algorithme CORDIC ont été comparées à celles obtenues d'un modèle analytique présenté par Kota dans [37]. La comparaison a démontré (1) que contrairement à certains modèles analytiques, la méthode peut trouver différentes soltutions pour différentes architectures, et (2) que les modèles analytiques peuvent restreindre les chemins de données à des tailles spécifiques, ce qui limite l'espace de solution à des solutions qui exigent un plus grand coût d'implantation.

La seconde adaptation apportée à la méthode a permis de déterminer simultanément, pour certaines applications spécifiques, les tailles des chemins de données

et la valeur des paramètres contenus dans les algorithmes. Cette modification est d'autant plus importante sachant que les paramètres contenus dans les algorithmes de traitement de signaux sont souvent déterminés empiriquement [55][45][40]. Cette adaptation permet d'ajuster spécifiquement la valeur des paramètres sous l'influence de la résolution virgule fixe de l'algorithme. Cette seconde adaptation a permis de déterminer avec succès les paramètres d'un décodeur à seuils itératifs dans [52] et de l'algorithme ELA pour le dé-entrelacement des images dans [21].

En résumé, ces recherches ont permis à l'auteur de cette thèse de présenter 4 articles de conférence [8][14][13][11], de publier 2 articles de revues [10][9] et de soumettre 2 autres articles de revues [15][12], en tant qu'auteur principal. D'autres recherches relatives à l'utilisation de la méthode avec des collaborateurs ont mené à la publication de 3 autres articles de conférence [19][52][21] et un autre article de revue [20].

Bien que plusieurs publications ont été effectuées, plusieurs autres projets de recherche peuvent être encore envisagés sur ce sujet, dont ceux-ci:

- Sachant que la méthode devient complexe (en terme de calcul) lorsqu'un grand nombre d'opérandes est analysé, déterminer si le nombre d'itérations peut être minimisé en regroupant les opérandes ayant des tailles similaires en une taille unique, et vérifier l'impact sur les solutions.

- Mesurer la sensibilité des solutions par rapport aux entrées présentées à l'algorithme, et déterminer s'il est possible de minimiser cette sensibilité lorsque la précision des résultats est déterminée à partir d'une distribution théorique lissée sur la distribution des erreurs observées aux sorties de l'algorithme.

- Comparer les différentes méthodes de détermination de la taille des chemins de données (DFG, statistiques et simulations) avec des exemples concrets.

- Sachant qu'à tout moment, les tailles et les valeurs des opérandes sont connues, il est possible de mesurer l'activité de chaque bit d'un chemin de données, et de caractériser plus précisément la consommation du modèle virgule fixe de l'algorithme analysé. Par conséquent, il serait possible d'optimiser la taille des chemins de données en fonction de cette dernière mesure.

- Afin de diminuer la sensibilité des solutions trouvées par la méthode aux entrées présentées à un algorithme, la taille du banc d'essai peut-être augmentée. Cependant l'augmentation de la taille du banc d'essai augmente le temps nécessaire à l'optimisation de la taille des chemins de données. Une détermination automatique des vecteurs d'entrées produisant une erreur maximale dans les résultats produits par l'algorithme permettrait (i) de réduire la taille du banc d'essai, (ii) de diminuer le temps nécessaire à l'optimisation et (iii) de s'assurer que les critères de précision spécifiés par l'usager sont respectés.

BIBLIOGRAPHIE

[1] AAMODT, T. Floating-Point to Fixed-Point Compilation and Embedded Architectural Support. Master's thesis, University of Toronto, 2001.

[2] ABDELRAZIK, M. A parallel multidimensional digital signal processing technique. *International Conference on Digital Processing of Signals in Communications* (1991), 268–271.

[3] ARCE, G., AND FONTANA, S. On the midrange estimator. *IEEE Trans. on Acoustics, Speech and Signal Processing vol. 36* (1988), 920–922.

[4] BOX, G., AND MULLER, M. A note on the generation of normal random deviates. *Annals of Mathmatical Statistics vol. 29* (1958), 610–611.

[5] BROWNRIGG, D. The weighted median filter. *Communications of the ACM vol. 27* (1984), 807–818.

[6] BURT, P., AND ADELSON, E. The laplacian pyramid as a compact image code. *IEEE Transactions on Communications* (1983), 532–540.

[7] CANTIN, M.-A., BLAQUIÈRE, Y., SAVARIA, Y., GRANGER, E., AND LAVOIE, P. Implementation of the fuzzy ART neural network for fast clustering of radar pulses. *IEEE International Symposium on Circuits and Systems vol. 2* (1998), 458–461.

[8] CANTIN, M.-A., BLAQUIÈRE, Y., SAVARIA, Y., LAVOIE, P., AND GRANGER, E. Analysis of quantization effects in a digital hardware implementation of a fuzzy ART neural network algorithm. *IEEE Int. Symposium on Circuits and Systems vol. 3* (2000), 141–144.

[9] CANTIN, M.-A., REGIMBAL, S., CATUDAL, S., AND SAVARIA, Y. An Unified Environment to Access Image Quality. *Journal of Circuits, Systems and Computers vol. 13*, 6 (2004).

[10] CANTIN, M.-A., AND SAVARIA, Y. An automatic world length determination method. *WSEAS Transaction on Information Science and Applications vol. 1*, 5 (2004), 1440–1448.

[11] CANTIN, M.-A., AND SAVARIA, Y. An automatic world length determination method. *Proceedings of the International Conference on Electronics, Control and Signal Processing* (2004).

[12] CANTIN, M.-A., SAVARIA, Y., BLAQUIÈRE, Y., AND BÉLANGER, N. Automatic Word Length and Number of Iterations Determination Method for the Hardware Optimization of Iteratives Algorithms. *Submitted to IEEE Transaction on Circuits and Systems* (2005).

[13] CANTIN, M.-A., SAVARIA, Y., AND LAVOIE, P. A Comparison of Automatic Word Length Optimization Procedures. *IEEE International Symposium on Circuits and Systems vol. 2* (2002), 612–615.

[14] CANTIN, M.-A., SAVARIA, Y., PRODANOS, D., AND LAVOIE, P. An automatic word length determination method. *IEEE International Symposium on Circuits and Systems vol. 5* (2001), 53–56.

[15] CANTIN, M.-A., SAVARIA, Y., PRODANOS, D., AND LAVOIE, P. A Metric for Automatic Word Length Determination of Hardware Datapaths. 5^{th} *Revision Submitted in March 2005 to IEEE Transaction on Computer-Aided Design* (2003).

[16] CARDINAL, C., HACCOUN, D., AND GAGNON, F. Iterative threshold decoding without interleaving for convolutional self-doubly orthogonal codes. *IEEE Transactions on Communications vol.IT-51*, 8 (2003), 1274–1282.

[17] CARPENTER, G., AND GROSSBERG, S. A massively parallel architecture for a self-organizing neural pattern recognition machine. *Computer Vision, Graphics and Image Processing vol. 37* (1987), 54–115.

[18] CAS STANDARDS COMMITTEE OF THE IEEE CIRCUITS AND SYSTEMS SOCIETY. IEEE Standard Specifications for the Implementations of 8x8 Inverse Discrete Cosine Transform, 1991.

[19] CATUDAL, S., CANTIN, M.-A., AND SAVARIA, Y. Performance Driven Validation Applied to Video Processing. *Proceedings of the 2004 WSEAS Conferences on Signal, Speech and Image Processing* (2004).

[20] CATUDAL, S., CANTIN, M.-A., AND SAVARIA, Y. Performance Driven Validation Applied to Video Processing. *WSEAS Transaction on Electronics vol. 1*, 3 (2004), 568–575.

[21] CATUDAL, S., CANTIN, M.-A., AND SAVARIA, Y. Parameters Estimation Applied to Automatic Video Processing Algorithms Validation. *Accepted to IEEE International Symposium on Circuits and Systems* (2005).

[22] CHANG, M., AND HAUCK, S. Precis: a design-time precision analysis tool. *IEEE Symposium on Field-Programmable Custom Computing Machines* (2002), 229–238.

[23] CHOI, H., AND BURLESON, W. P. Search-based wordlength optimization for VLSI/DSP synthesis. *VLSI Signal Processing VII* (1994), 198–207.

[24] CLAESEN, L., CATTHOOR, F., LANNEER, D., GOOSSENS, G., NOTE, S., MEERBERGEN, J. V., AND MAN, H. D. Automatic Synthesis of Signal Processing Benchmark using the CATHEDRAL Silicon Compilers. *Proceedings of IEEE Custom Integrated Circuits Conference* (1988), 14.7.1–14.7.4.

[25] CMAR, R., RIJNDERS, L., SCHAUMONT, P., VERNALDE, S., AND BOLSENS, I. A methodology and design environment for DSP ASIC fixed point refinement design. *Automation and Test in Europe Conference and Exhibition* (1999), 271–276.

[26] CROZIER, S. Performance and Complexity of Discrete-Time Frequency Estimation Algorithms. *17th Biennial Symposium on Communications, Queen's University, Ontario, Canada* (1994).

[27] ESKICIOGLU, A., AND FISHER, P. Image quality measures and their performance. *IEEE Transactions on Communications vol. 43*, 12 (1995), 2959–2965.

[28] FIORE, P., AND LEE, L. Closed-form and real-time wordlength adaptation. *Proceedings of the IEEE International Conference on Acoustics, Speech and Signal Processing vol. 4* (1999), 1897–1900.

[29] HAMPSON, G. A. *Implementing Multi-Dimensional Digital Hardware Beamformers.* PhD thesis, Monash University, Department of Robotics and Digital Technology, 1997.

[30] HAN, K., EO, I., KIM, K., AND CHO, H. Numerical word-length optimization for CDMA demodulator. *IEEE International Symposium on Circuits and Systems vol. 4* (2001), 290–293.

[31] KALMAN, R. A New Approach to Linear Filtering and Prediction Problems. *Transactions of the ASME, Journal of Basing Engineering vol. 82* (1960), 34–35.

[32] KEDING, H., WILLEMS, M., COORS, M., AND MEYR, H. FRIDGE: a fixed-point design and simulation environment. *Design, Automation and Test in Europe* (1998), 429–435.

[33] KIM, S., KUM, K.-I., AND SUNG, W. Fixed-Point Optimization Utility for C and C++ Based Digital Signal Processing Programs. *IEEE Transaction on Circuits and Systems II vol. 45*, 11 (1998), 1455–1464.

[34] KIM, S., AND SUNG, W. Fixed-Point Error Analysis and Word Length Optimization of 8x8 IDCT Architectures. *IEEE Transaction on Circuits and Systems for Video Technology vol. 8*, 8 (1998), 935–940.

[35] KIRKPATRICK, S., GELATT, C., AND VECCHI, M. Optimization by simulated annealing. *Science 220* (1983), 671–680.

[36] KORN, I. *Digital Communications.* Van Nostrand Reinhold, 1985.

[37] KOTA, K., AND CAVALLARO, J. R. Numerical Accuracy and Hardware Tradeoffs for CORDIC Arithmetic for Special-Purpose Processors. *IEEE Transactions on Computers vol. 42,* 7 (1993), 769–779.

[38] LAFRANCE, L.-P., CANTIN, M.-A., SAVARIA, Y., SUNG, S., AND LAVOIE, P. Architecture and performance characterization of hardware and software implementations of the crozier frequency estimation algorithm. *IEEE International Symposium Circuits and Systems vol. 4* (2002), 823–826.

[39] LANE, T. G. The independant jpeg group's software, release 5b, 1990-1995. Archive site: *ftp.uunet.*

[40] LEE, H., PARK, J., BAE, T., CHOI, S., AND HA, Y. Adaptive scan rate up-conversion system based on human visual characteristics. *IEEE Transactions on Consumer Electronics vol. 46,* 4 (2000), 999–1006.

[41] LEE, S.-G., AND LEE, D.-H. A motion-adaptive de-interlacing method using an efficient spatial and temporal interpolation. *IEEE Transactions on Consumer Electronics vol. 49,* 4 (2003), 1266–1271.

[42] LIM, I.-T., AND BAHN, J. Optimal wordlength determination of AC-3 decoding hardware based on fixed-point analysis and simulations of AC-3 algorithm. *IEEE Workshop on Signal Processing* (1997), 301–310.

[43] LIM, J. *Two-Dimensional Signal and Image Processing.* Prentice Hall, 1990.

[44] LOAN, C. V. *Introduction to Computational Science and Mathematics.* Jones and Bartlett Publishers, 1996.

[45] LOISEAU, L. Méthode de conception pour la réutilisation et optimisation architecturale appliqueées à un réducteur de bruit vidéo. Master's thesis, Ecole Polytechnique de Montréal, 2001.

[46] MCDONNELL, M. Box-filtering techniques. *Computer Graphics and Image Processing vol. 17* (1981), 65–70.

[47] MIYAZAKI, T., NISHITANI, T., EDAHIRO, M., AND ONO, I. DCT/IDCT processor for HDTV developed with DSP silicon compiler. *Journal of VLSI Signal Processing vol. 5* (1993), 39–47.

[48] OKADA, K., YAMADA, A., AND KAMBE, T. Hardware algorithm optimization using Bach C. *IEICE Transactions on Fundamentals of Electronics, Communications and Computer Sciences vol.E85-A*, 4 (2002), 835–841.

[49] OSELI, D., MRAZ, M., AND ZIMIC, N. Design considerations of hardware based fuzzy controllers. *WSEAS Transaction on Electronics vol. 3* (2004), 811–816.

[50] PALNITKAR, S. *Design Verification with e.* 2002.

[51] PAPPAS, T., AND SAFRANEK, R. *Perceptual criteria for image quality evaluation.* Handbook of Image and Video Processing. Academic Press, 2000.

[52] PROVOST, G., CANTIN, M.-A., SAWAN, M., CARDINAL, C., Y.SAVARIA, AND HACCOUN, D. Fast parameters optimization of an iterative decoder using a configurable hardware accelerator. *accepted to the International Symposium on Circuits and Systems* (2005).

[53] RAND, W. Objective Criteria for the Evaluation of Clustering Methods. *Journal of the American Statistical Association vol. 66*, 336 (1971), 846–850.

[54] SHIUE, W.-T. Low power vlsi design: An optimal binding algorithm in high-level synthesis using integer linear programming. *Proceedings of the World*

Multiconference on Systemics, Cybernetics and Informatics vol. 15 (2002), 458–461.

[55] SMITH, S., AND BRADY, J. SUSAN - A New Approach to Low Level Image Processing. *International Journal of Computer Vision vol. 23* (1997), 45–78.

[56] SPEITEL, M., AND NIEMANN, B. SystemC design language for development of ASICs and systems. *Elektronik vol. 50*, 13 (2001), 78–83.

[57] SUNG, W., AND KUM, K. Simulation-based word-length optimization method for fixed-point digital signal processing systems. *IEEE Transaction on Signal Processing vol. 43*, 12 (1995), 3087–3090.

[58] SYNOPSYS INC. *Press Release: Synopsys Accelerates System-Level C-Based DSP Design With CoCentric Fixed-Point Designer Tool*, 2000.

[59] TUKEY, J. *Exploratory Data Analysis.* Addison-Wesley, 1971.

[60] VOLDER, J. The CORDIC Trigonometric Computing Technique. *IRE Transaction on Electronic Computers vol.EC-8*, 3 (1959), 330–334.

[61] WADEKAR, S. Accuracy Sensitive Word-length Selection for Algorithm Optimization. *Proceedings of the International Conference on Computer Design: VLSI in Computers and Processors* (1998), 54–61.

[62] WALLACE, G. The jpeg still picture compression standard. *Communications of the ACM vol. 34*, 4 (1991), 30–44.

[63] WANG, Z., AND BOVIK, A. A universal image quality index. *IEEE Signal Processing Letters vol. 9*, 3 (2002), 81–84.

[64] WANG, Z., AND BOVIK, A. Why is image quality assessment so difficult? *IEEE International Conference on Acoustics, Speech, and Signal Processing vol. 4* (2002), 3313–3316.

[65] WEBER, A. G. The usc-sipi image database: Version 5. Master's thesis, University of Soutern California, 1997.

[66] XIANGZHEN, Q. Cache performance and algorithm optimization. *Proceedings of the High Performance Computing on the Information Superhighway HPC Asia* (1997), 12–17.

[67] YAMASHITA, H., YASUURA, H., EKO, F. N., AND YUN, C. Variable Size Analysis and Validation of Computation Quality. *Proceedings of Workshop on High-Level Design Validation and Test* (2000), 95–100.

[68] YLI-KAAKINEN, J., AND SARAMAKI, T. An efficient algorithm for the design of lattice wave digital filters with short coefficient wordlength. *International Symposium on Circuits and Systems vol. 3* (1999), 443–448.