Optimisation automatique d'un Data Flow Graph à l'aide du formalisme TED. Cas d'étude: La Transformée de Fourier Discrète.

J. Guillot[1], E. Boutillon[1], M. Ciesielski[2] D. Gomez-Prado[2]

[1]Laboratoire LESTER
Université de Bretagne Sud, 56321 LORIENT

[2]VLSI CAD LAB
University of Massachusetts Amherst, MA 01003 USA

jeremie.guillot, emmanuel.boutillon@univ-ubs.fr, ciesiel, dgomez@ecs.umass.edu

Résumé − Ce papier décrit une méthode d'optimisation de transformées de type traitement du signal en effectuant des transformations à hauts niveaux de types factorisations et élimination de sous expressions communes basées sur les Taylor Expansion Diagrams. Cette méthodologie en cours d'intégration dans le démonstrateur TEDify [11] permet déjà d'obtenir des résultats performant et originaux du point de vue optimisation. Intégrée à un outil de synthèse elle permettra de réduire de façon considérable le temps de conception des circuits numériques en permettant au concepteur d'atteindre plus facilement les contraintes imposées: Consommation, surface, débit.

Abstract − An optimisation scheme of usual DSP transforms using high level transformations is proposed. These transformations are based on factorizations and common subexpression eliminations process using a Taylor Expansion Diagram representation. The implementation of this methodology is currently under development and validation phases but, it already shows the the performances and the originalities of the generated solutions.

1 Introduction

Un des enjeux des compilateurs et des outils de synthèse repose sur leurs capacités à déduire de la description d'entrée le flot de calculs permettant de réaliser la fonctionnalité spécifiée par l'utilisateur. Ces outils [1] [2] [3] [6] dérivent un arbre syntaxique du code écrit par le concepteur afin de réaliser une passe de simplification triviale permettant d'optimiser le calcul avant compilation ou synthèse (élimination de code mort, déplacement de code, élimination des redondances). En revanche, ces outils ne tirent pas parti de toutes les potentialités de simplification évoluées qui peuvent exister dans la spécification d'entrée.

Pour palier à ce défaut, différentes équipes proposent des outils dédiées aux principales transformées utilisées en traitement du signal. Ces outils sont utilisables pour une classe prédéfinie d'algorithmes. Ils tirent partie de la connaissance à priori des optimisations possibles pour générer la solution de coût minimal. La principale restriction de ces outils, est que leur domaine d'utilisation, est limité par construction.

Le projet décrit dans cet article est différent. Nous essayons de reproduire les résultats des outils dédiés mais cette fois-ci, sans utiliser de connaissance a priori sur les transformées en entrée. Notre ambition est d'obtenir un outil réellement générique et performant. Notre approche est basée sur l'utilisation du formalisme des Taylor Expansion Diagrams (TEDs) [7] et sa mise en oeuvre dans l'outil TEDify [11] co-développé par l'Université du Massachusetts et l'Université de Bretagne Sud. Les TEDs sont un formalisme récent permettant de représenter un ensemble de fonctions polynomiales multivariées sous la forme d'un graphe ordonné canonique. Ce formalisme a été proposé initialement par le Professeur Ciesielski dans le contexte de la vérification de systèmes numériques (montrer que deux circuits (ou descriptions) différents ont la même fonctionnalité). Ensuite, nous avons montré que ce formalisme pouvait être utilisé dans le domaine de la synthèse d'architecture [10] et la compilation. Notre objectif est de s'affranchir du mode de description du concepteur et de générer automatiquement un graphe de calcul plus approprié par rapport aux contraintes de l'application. Un premier pas a été fait dans [8] où une méthode de factorisation automatique des sous expressions communes est présentée (méthode "CSE", pour Common Subexpression Elimination). Cette méthode est mise en oeuvre dans TEDify. Elle permet, par exemple de retrouver automatiquement le graphe factorisé d'une transformée de Wash-Hadamard à partir de la description du produit matrice vecteur en entrée. Dans [9], nous montrons comment dériver automatiquement un graphe flot de calcul optimisé à partir d'un TED. Toutefois, l'utilisation de TEDify dans sa forme actuelle ne permet pas de retrouver la forme "optimale" de la Transformée de Fourier Rapide (TFR). En effet, seuls les premiers étages de radix-2 sont extraits, les factorisations nécessitant la décomposition des coefficients de la Transformée de Fourier Discrète (TFD) comme produit d'autres coefficients ne sont pas reconnues dans l'outil actuel.

Dans ce papier, nous présentons une nouvelle règle de

factorisation ainsi qu'une méthode de parcours de graphe permettant de trouver les candidats à cette factorisation. Nous montrons que, combiné avec une règle de priorité pour choisir l'ordre des factorisations, il est possible de retrouver automatiquement la TFR à partir de la description initiale de la TFD comme produit matrice vecteur.

Le reste de l'article est divisé en quatre parties. Dans la première partie, nous rappelons brièvement le principe de construction d'un TED ainsi que l'extraction d'expression commune. La seconde partie décrit les nouvelles règles d'optimisations. Nous montrons brièvement les résultats de l'application de ces algorithmes sur la TFR. Enfin, nous concluons cet article en présentant les limites actuelles de notre approche et en décrivant les travaux futurs.

2 Taylor Expansion Diagrams

Les Taylor Expansion Diagrams (TEDs) sont issues de la décomposition d'expression arithmétique en développement de Taylor. Le développement en Série de Taylor permet de représenter une fonction infiniment dérivable par un polynôme construit suivant l'équation 1.

$$f(x) = \sum_{n=0}^{\infty} \frac{f^{(n)}(a)}{n!}(x-a)^n \tag{1}$$

Par souçis de concision, la figure 1 servira de support à l'explication sommaire d'un TED. Cette figure représente le TED des fonctions polynomiales $F1$ et $F2$ avec pour ordre de variables : d, c, a, b. Chaque noeud du graphe est associé à une variable X. Ce noeud représente un polynôme $P(X, ...)$ de la variable X et des variables suivantes. On interprète le sous-graphe pointé par l'arc pointillé (arc d'ordre 0) partant du noeud X comme étant le polynome obtenu avec $X = 0$. Le sous-graphe pointé par l'arc plein (arc d'ordre 1) est le polynome obtenu en dérivant $P(X, ...)$ par rapport à X avec $X = 0$. Par exemple, du noeud c, il n'y a pas d'arc pointillé, ce qui implique que $P_c(c = 0, a, b) = 0$. Du noeud c un arc plein atteind le noeud a de polynome $P_a(a, b)$. On en déduit que $P_c(c, a, b) = c.P_a(a, b)$. En procédant récursivement des premières variables aux dernières variable, on en déduit $F1 = c \times (a + b)$ et $F2 = d \times (a + b)$. Le lecteur pourra trouver dans [7] une description exhaustive des TEDs.

Une redondance de calcul dans un TED est repérée à chaque fois qu'il existe plus d'un arc parent arrivant à un noeud non trivial comme illustré pour le noeud a de la figure 1. Chaque fois qu'une redondance est détectée, une variable intermédiaire regroupant le sous-graphe est générée afin de simplifier le calcul (ici, $s = a+b$). Le processus de factorisation CSE repose sur la détection (en utilisant un ordre de variable approprié) et la suppression de ces expressions partagées [8] [10]. Toutefois, comme indiqué dans l'introduction, l'algorithme CSE ne permet pas d'obtenir une TFR à partir d'une TFD. D'autres règles doivent donc être rajoutées.

La première de ces règles concerne directement la partie CSE. La version CSE implémentée à ce jour dans TEDify ne permet pas de détecter systématiquement toutes les expressions communes. En effet, pour qu'une sous-expression commune soit identifiée, il faut que les variables associées au polynôme commun soient situées en fin de la liste des variables du TED. Tout les ordres n'étant pas testés, des termes communs peuvent ainsi échapper à la version actuelle de TEDify. Dans le cas du produit de deux variables réutilisés dans plusieurs expressions polynomiales, il est toutefois assez facile de résoudre le problème : il suffit de détecter dans le graphe qu'il existe plusieurs « chemins multiplicatifs »entre deux variables.

3 Nouvelles règles de simplification

Après avoir décrit une nouvelle règle de factorisation, nous décrirons une méthode permettant d'identifier les factorisations possibles et laquelle effectuer en premier. L'application itérative de cette méthode permettra de factoriser efficacement le TED.

3.1 Factorisation

Dans un TED la factorisation s'intéresse à la décomposition d'un coefficient de Taylor en une combinaison linéaire d'autres coefficients avec pour fonction objectif de réduire globalement le nombre d'opérations nécessaires aux calculs. Ce nombre ϕ est défini par le nombre d'arcs entre noeuds (hors noeud terminal « un »). Ainsi, ϕ_{MPY} et ϕ_{ADD} correspondant respectivement, aux nombres de multiplieurs et d'aditionneurs nécessaires au calcul sont obtenu à partir du nombre d'arcs non triviaux d'ordre un et d'ordre zéro (les arcs triviaux sont ceux arrivants au noeud 1).

Considérons maintenant le TED de la figure 2 pour lequel $\phi_{MPY} = 4$ et $\phi_{ADD} = 2$ et correspondant au jeu d'équations 2 :

$$\begin{cases} Y1 &= A \times X1 + B \times X2 \\ Y2 &= C \times X1 + D \times X2 \end{cases} \tag{2}$$

En considérant que A, B, C et D sont des constantes non nulles et que $X1$ et $X2$ sont des variables, on obtient alors l'expression d'un produit matrice vecteur 2x2. Il est possible, en choisissant arbitrairement $X1$ comme variable de réference, de réécrire $Y1$ et $Y2$ sous la forme 3 :

$$\begin{cases} Y1 &= A \times (X1 + \frac{B}{A} \times X2 \\ Y2 &= C \times (X1 + \frac{D}{C} \times X2) \end{cases} \tag{3}$$

Cette factorisation permet de diminuer la fonction de coût si $(\frac{B}{A})^2 = (\frac{D}{C})^2$. En effet, on obtient alors $\alpha = \frac{B}{A} = \varepsilon\frac{C}{D}$ avec $\varepsilon \in \{-1; 1\}$. La forme 3 devient alors :

$$\begin{cases} Y1 &= A \times (X1 + \alpha \times X2 \\ Y2 &= C \times (X1 + \varepsilon\alpha \times X2) \end{cases} \tag{4}$$

Cette forme factorisée ne contient plus que trois multiplieurs et, si $\varepsilon = 1$, plus que un unique additionneur (dans ce cas, la matrice est dégénérée). Ainsi, l'égalité $(\frac{B}{A})^2 = (\frac{D}{C})^2$ permet de diminuer le nombre d'opération nécessaire au calcul de $Y1$ et $Y2$.

Si nous appliquons cette propriété au cas où $A = \pm D$ et $B = \pm C$, alors $A^2 = -B^2$ implique aussi une factorisation. Cette propriété est utilisée dans la suite pour permettre de transformer une TFD en TFR. La figure 3 donne le TED de la TFD 8 avant factorisation et on peut noter que les différentes variables en jeux forment un cycle dans le graphe. Ce cycle se caractérise par :

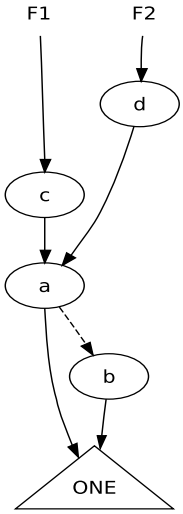

FIG. 1 – Exemple : $F1 = c \times (a + b)$ et $F2 = d \times (a + b)$

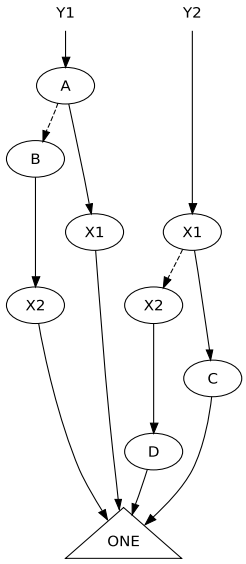

FIG. 2 – Exemple : TED des fonctions $Y1 = A \times X1 + B \times X2$ et $Y2 = C \times X1 + D \times X2$

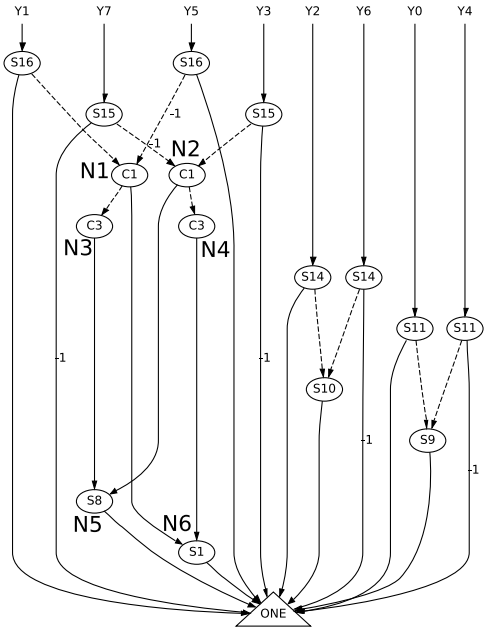

FIG. 3 – TED de la TFD après CSE

1. Une longueur totale de 6 arcs.
2. Des arcs additifs entre les constantes.
3. Des arcs multiplicatifs entre les constantes et les variables.

Dans cette figure, le cycle apparait explicitement. Il est bien plus difficile d'identifier visuellement de tels cycles dans un TED complexe. Toutefois, partant de deux constantes, il est possible de les détecter automatiquement en un temps raisonnable par un parcours de graphe approprié. Dans le cas où plusieurs factorisations sont possibles, celle apparaissant dans le plus d'expressions de sortie est alors choisie en premier.

4 Application à la transformée de Fourier Discrète

Après avoir rappelé les équations de la TFD, nous décrirons en détail comment appliquer les algorithmes de CSE et de factorisation au TED de la TFD pour obtenir une TFR.

4.1 Rappel sur la TFD

Nous avons utilisé cette approche dans le cadre du calcul de la Transformée de Fourier Discrète (TFD). A titre d'exemple nous nous basons sur la TFD de taille 8 (TFD8) suivant une décimation en temps. La matrice des coefficients de Fourier est obtenue à l'aide de la formule :

$$Y_k = \sum_{n=0}^{7} e^{\frac{-2\pi i}{8} nk} X_n \qquad k = 0, \ldots, 7$$

Il est possible de réécrire cette expression par :

$$Y_k = \sum_{n=0}^{7} (-1)^{((nk) mod 4)} C_{((nk) mod 4)} X_n \text{ avec } C_n = e^{\frac{-2\pi i n}{8}}$$

Le TED initial de la TFD8 est donnée figure 4.

4.2 Factorisation de la TFD en TFR

L'étape préliminaire à la factorisation consiste à détecter les couples de constantes vérifiant $C_i^2 = -C_j^2$. Une telle liste peut être réalisée automatiquement en calculant le carré de chaque constante puis, en extrayant les couples vérifiant cette propriété. Elle peut aussi être donnée comme à priori à l'algorithme de factorisation. Une fois l'étape préliminaire effectuée, l'algorithme de factorisation devient :

1. Tant que le TED est modifiée
2. Faire l'algorithme CSE
3. Remonter toutes les termes additifs purs
4. Détecter l'ensemble des factorisations possibles
5. Factoriser les polynômes dont les cycles sont de cardinalité maximale.

Nous indiquons ci-dessous les sous expressions obtenues après la première application de l'algorithme CSE pour la TFD8.

```
S1  = X1-X5   S2 = X5+X1    S3 = X0-X4
S4  = X4+X0   S5 = X2-X6    S6 = X6+X2
S7  = X7+X3   S8 = X3-X7    S9 = S6+S4
S10 = S4-S6  S11 = S7+S2   S12 = S2-S7
S13 = C2*S5  S14 = C2*S12  S15 = S13-S3
S16 = S13+S3
```

L'algorithme d'analyse du TED permet de détecter le cycle (N1,N6,N4,N2,N5,N3,N1) correspondant au jeu d'équations 2 avec $A = D = C1$ et $B = C = C3$ et effectue alors le calcul des polynômes factorisée sous la forme du jeu d'équations 4.

Comme le montre la figure 5, le graphe est fortement simplifié après factorisation à l'aide de TEDify. Une fois simplifié, le calcul de la DFT8 est réalisé en 24 additions et 5 multiplications (contre respectivement 56 et 64 dans

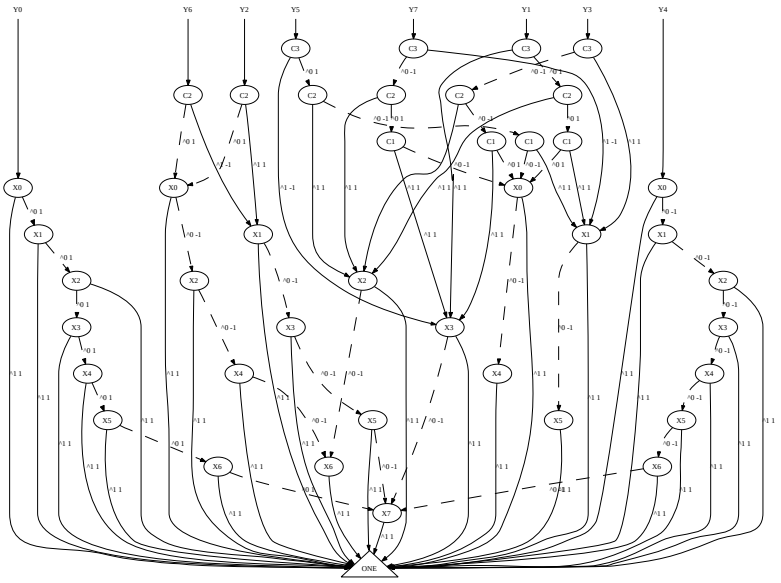

FIG. 4 – TED de la TFD8 avant optimisation par TEDify

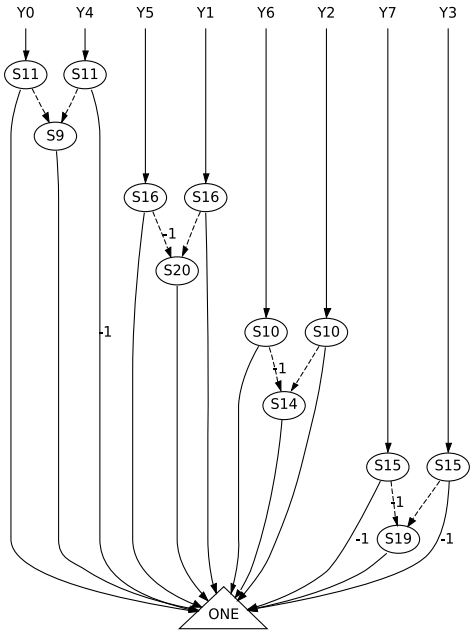

FIG. 5 – TED de la TFD8 après factorisation

le cas du produit matrice vecteur) et montre ainsi la pertinence de ce formalisme. Ce résultat peut être encore amélioré en décomposant les complexes suivant leurs parties réelles et imaginaires mais cela n'est pas le cadre de cette étude. Il est en revanche interessant de noter que les opérations restantes dans le TED représentent les opérations du dernier étage de papillons de la TFR.

Pour le moment, cet algorithme de factorisation a été testé à la main avec succès sur des TFD de taille 8 et 16 et, bien que non formellement démontré, nous sommes confiant dans le fait que l'algorithme est applicable aux tailles supérieures (64, 128,...). L'étape suivante, pour valider notre approche, est d'intégrer cet algorithme dans l'outil TEDify et d'évaluer le temps de calcul pour effectuer de telles optimisations.

5 Conclusions

Par l'utilisation d'un formalisme orginal, nous avons montré que les TEDs, permettent de retrouver la factorisation automatique d'une TFD en une TFR pour des tailles égales à des puissance de deux. L'avantage de l'approche proposée réside dans sa généricité. En effet, les méthodes de simplification reposent sur les liens existants entre les variables symboliques et non pas sur la taille ou le type des transformées traitées. Les résultats présentés ici bien que préliminaires, nous ont permis de retrouver les algorithmes de J. Cooley et J. Tuckey publiés en 1965 pour des TFD de taille 8 et 16.

Nous évaluons actuellement les règles présentées dans cet article afin de voir si elles permettent de factoriser efficacement d'autres types de transformées : Transformée en cosinus discrète, TFD et factorisation split radix, TFD pour des tailles quelconques, filtre polyphase, etc. Eventuellement, les règles présentées ici seront à compléter afin d'être le plus générique possible.

L'étape suivante sera l'intégration de ces règles dans l'outil TEDify afin d'obtenir un véritable outil d'optimisation générique et universel. Cet outil poura alors être utilisé en amont de la compilation ou de la synthèse d'architecture pour réécrire la description d'entrée de façon optimisée par rapport aux contraintes de l'application.

Références

[1] S. Gupta et al. *Spark : A high-level synthesis framework for applying parallelizing compiler transformations*, Intl. Conf. on VLSI Design, 2003.

[2] K. Wakabayashi *C-based synthesis experiences with a behavior synthesizer : Cyber* DATE-1999.

[3] Mentor Graphics Corp, Wilsonville, USA *Catapult C Synthesis*

[4] M. Puschel, et al *Spiral : Code Generation for DSP Transforms*, Proceedings of the IEEE special issue on « Program Generation, Optimization, and Adaptaion », Vol.93, No.2, 2005, p.232-275

[5] Frigo et al. *The Design and Implementation of FFTW3* Proceedings of the IEEE special issue on « Program Generation, Optimization, and Adaptaion », Vol.93, No.2, 2005, p.216-231

[6] GNU Free Software Foundation *GCC*

[7] Ciesielski et al. *Taylor Expansion Diagrams : A Compact, Canonical Representation with Applications to Symbolic Verification*,DATE-2002 p285-289

[8] Guillot et al. *Efficient Factorization of DSP Transforms using Taylor Expansion Diagrams* DATE-2006 p754,756

[9] Ciesielski et al. *Data-Flow Transformations Using Taylor Expansion Diagrams* DATE-07

[10] M. Ciesielski, S. Askar, E. Boutillon, J Guillot *Behavioral transformations for hardware synthesis and code optimization based on taylor expansion diagrams* US Utility patent, USSN11/292,493 filed 12/02/2005 and PCT application PCT/US05/43860, filed 12/03/2005

[11] http ://tango.ecs.umass.edu/TED/Doc/html/