

Système de contraintes pour DSP C6x

Lionel Lacassagne

1 Introduction

Le but de ces systèmes de contraintes est d'estimer la cadence de calcul d'un algorithme, en prenant en compte (système *In Order*) ou non (système *Out of Order*) le chronogramme de l'algorithme.

La table 1 donne les cadences et latences des principales instructions utilisées pour le codage des algorithmes. Pour information, les **SHIFT** s'exécutent sur les mêmes unités fonctionnelles que les **ADD** et **SUB**. En fait il est possible de faire 4 opérations arithmétiques et logiques sur un C64, car il y a 4 ALU, avec un maximum de 2 **SHIFT** par cycle. Pour rappel, il est possible de faire 2 opérations arithmétiques supplémentaires, si on ne fait pas d'accès mémoire. Pour le système de contrainte, on utilisera la valeur 4 pour la partie Arithmétique/Addition.

type d'instructions	exemple d'instruction	cadence	latence
I : instruction	toute <i>instruction</i>	8	1 à 5
E/S : entrée/sortie	LOAD / STORE	2	5 / 1
M : Multiplication	MUL / DOTP2	2	2 / 4
A : Arithmétique	ADD et SUB	4	1

TAB. 1 – cadence et latence des instructions C6x

Ces systèmes de contraintes vont donc permettre d'évaluer un nombre de cycles par point de calcul (*c_{pp}*), avant optimisation, et après. Cette évolution du *c_{pp}* permettra de comparer les différentes implantation.

2 Système de contraintes *Out of Order*

Le système de contraintes *Out of Order* ne prend en compte que le nombre d'instruction de chaque type, présent dans un algorithme, et pas le scheduling lié à l'algorithme. Ce système n'est donc pas directement sensible aux techniques de déroulage de boucle (Loop Unrolling Interne ou Externe), mais peut être sensible à ses conséquences (diminution du nombre d'accès mémoire via Loop Unrolling ou Rotation de Registres).

2.1 Système de contraintes

Pour tous les types i d'instructions de l'ensemble I, ES, M, A , soient les notations suivantes :

- n_i , le nombre d'instructions présentes dans le code pour chaque type,
- c_i , le nombre de cycles minimum nécessaire pour exécuter, pour chaque type, les n_i instructions
- d_{imax} , la densité maximale d'instruction pouvant être lancée à chaque cycle, pour chaque type d'instruction, c'est à dire la cadence,
- c , le nombre minimal de cycle pour exécuter l'algorithme, sous l'hypothèse qu'il existe un ordonnancement (*scheduling*) des instructions pour y arriver. Comme c peut ne pas être entier, deux versions sont calculées c_{calc} qui est le résultat du calcul et c_{prat} qui est le nombre de cycles obtenu en pratique, en arrondissant c_{calc} ,
- p le nombre de points calculés en une itération de boucle (très souvent $p = 1$)
- c_{pp} qui est le nombre de cycles par point de calcul (deux versions $c_{pp_{calc}}$ et $c_{pp_{prat}}$).

Les formules de calculs sont :

$$\begin{aligned}
 c_i &= \frac{n_i}{d_{imax}} \\
 c_{calc} &= \max(c_i) \\
 c_{prat} &= \lceil \max(c_{calc}) \rceil \\
 cpp_{calc} &= c_{calc}/p \\
 cpp_{prat} &= c_{prat}/p
 \end{aligned}$$

2.2 Exemples

Soit à implanter le filtre

$$y(n) = x^2(n) + x^3(n-1)$$

En fonction du codage on obtient différents codes et différents systèmes de contraintes :

2.2.1 Implantation basique

```

for(i=0; i<n;i++){
  Y[i] = X[i]*X[i] + X[i-1]*X[i-1]*X[i-1];
}

```

TAB. 2 – implantation basique

type d'instructions	n_i	d_{imax}	c_i
I	10	8	$10/8 = 1.25$
E/S	6	2	$6/2 = 3$
M	3	2	$3/2 = 1.5$
A	1	4	$1/4 = 0.25$

TAB. 3 – système *Out of Order* pour le code basique

Pour le code (table 2), le système de contraintes (table 3) donne $c_{calc} = 3$ et $c_{prat} = 3$. Comme $p = 1$, $cpp_{calc} = 3$ et $cpp_{prat} = 3$. Le facteur limitant est le nombre d'accès mémoire (6 pour 3 multiplications).

2.2.2 Implantation registre

```

for(i=0; i<n;i++){
  x0 = X[i];
  x1 = X[i-1];
  y = x0*x0 + x1*x1*x1;
  Y[i] = y;
}

```

TAB. 4 – implantation "registre"

Pour le code (table 4), le système de contraintes (table 5) donne $c_{calc} = 1.5$ et $c_{prat} = 2$. Comme $p = 1$, $cpp_{calc} = 1.5$ et $cpp_{prat} = 2$. Le facteur limitant est à la fois le nombre d'accès mémoire et le nombre de multiplications.

type d'instructions	n_i	d_{imax}	c_i
I	7	8	$7/8 = 0.875$
E/S	3	2	$3/2 = 1.5$
M	3	2	$3/2 = 1.5$
A	1	4	$1/4 = 0.25$

TAB. 5 – système *Out of Order* pour le code Register

```

x1 = X[-1] ;
for(i=0; i<n; i++){
  x0 = X[i] ;
  y = x0*x0 + x1*x1*x1 ;
  x1 = x0 ;
  Y[i] = y ;
}

```

TAB. 6 – implantation avec optimisation Rotation de Registres

2.2.3 Implantation avec Rotation de Registres

Pour le code (table 6), le système de contraintes (table 7) donne $c_{calc} = 1.5$ et $c_{prat} = 2$. Comme $p = 1$, $c_{pp_{calc}} = 1.5$ et $c_{pp_{prat}} = 2$. Le facteur limitant est maintenant le nombre de multiplication. La vitesse de calcul ne dépend plus du nombre d'accès mémoire, mais uniquement du nombre de calcul, qui est la situation idéale.

2.2.4 Remarques

- pour un même algorithme, il existe plusieurs codages possibles avec différents niveaux d'optimisation, donnant différents résultats pour le système de contraintes,
- le système ne garanti pas qu'un scheduling correct existe,
- si le nombre de cycle est fractionnaire, arrondir au nombre supérieur.

3 Système de contraintes *In Order*

Ce modèle utilise un chronogramme pour réaliser une estimation plus précise du nombre de cycle nécessaire, et du nombre de duplication de code à réaliser. Pour tous les types i d'instructions de l'ensemble I, ES, M, A , soient les notations suivantes :

- n_i , le nombre d'instructions présentes dans le code pour chaque type,
- c , le nombre de cycles d'une itération de la boucle (ie la longueur du chronogramme),
- d_{imax} , la densité maximale d'instruction,
- d_i , la densité moyenne par type d'instruction,
- k_i , le facteur de duplication de code, en ne considérant qu'un type d'instruction,
- k_{calc} , le facteur de duplication effectif du code, sous l'hypothèse qu'il existe un ordonnancement (*scheduling*) des instructions pour y arriver.
- k_{prat} , facteur de duplication pratique,
- p le nombre de points calculés en une itération de boucle (très souvent $p = 1$)
- c_{pp} qui est le nombre de cycles par point de calcul (deux versions $c_{pp_{calc}}$ et $c_{pp_{prat}}$).

Nous avons :

$$\begin{aligned}
 d_i &= \frac{n_i}{c} \\
 k_i &= \frac{d_{imax}}{d_i} = \frac{d_{imax}}{n_i/c} \\
 k_{calc} &= \min(k_i) \\
 k_{prat} &= \lfloor k_{calc} \rfloor
 \end{aligned}$$

type d'instructions	n_i	d_{imax}	c_i
I	6	8	$6/8 = 0.75$
E/S	2	2	$2/2 = 1$
M	3	2	$3/2 = 1.5$
A	1	4	$1/4 = 0.25$

TAB. 7 – système *Out of Order* pour le code Rotation de Registres

$$cpp_{calc} = c/p/k_{calc}$$

$$cpp_{prat} = c/p/k_{prat}$$

3.0.5 Implantation basique

0	LL	$X[i] X[i]$	0	LL	$X[i+1] X[i+1]$
1	LL	$X[i+1] X[i+1]$	1	LL	$X[i] X[i]$
2	L	$X[i+1]$	2	L	$X[i+1]$
3			3		
4			4		
5	x	$X[i] * X[i]$	5	x	$X[i+1] * X[i+1]$
6	x	$X[i+1] * X[i+1]$	6	x	$X[i] * X[i]$
7			7	x	$X[i+1] * X[i+1] * X[i+1]$
8	x	$X[i+1] * X[i+1] * X[i+1]$	8		
9			9	+	
10	+	$X[i] * X[i] + X[i+1] * X[i+1] * X[i+1]$	10	S	$X[i] * X[i] + X[i+1] * X[i+1] * X[i+1]$
11	S				

FIG. 1 – chronogrammes associés à l'implantation basique

Le chronogramme basique associé au code est construit en prenant en compte les contraintes architecturales du C6x (un maximum de 2 accès mémoire, de 2 multiplications et de 4 additions par cycle) et en “lançant et en enchaînant les instructions au plus tôt”. Néanmoins il faut faire attention à prendre en compte le plus long chemin associé à graphe de dépendance de donnée (figure 2). Ainsi sur la figure 1, le chronogramme de gauche ne prend pas en compte le plus long chemin, alors que celui de droit le prend en compte.

type d'instructions	n_i	d_{imax}	c	d_i	k_i
I	10	8	12	$10/12 = 0.83$	$8/0.83 = 9.6$
E/S	6	2	12	$6/12 = 0.5$	$2/0.5 = 4$
M	3	2	12	$3/12 = 0.25$	$2/0.25 = 8$
A	1	4	12	$1/12 = 0.08$	$2/0.08 = 48$

TAB. 8 – système de contraintes *In Order* associé au code basique

Sans prendre en compte le plus long chemin, pour le code (table 2), le système de contraintes (table 9) donne des facteurs de duplication calculé et pratique identiques : $k = 4$, et le cpp vaut $cpp = 3$ ($p = 1$). En prenant en compte le plus long chemin dans du graphe (figure 1, chronogramme de droite), le système de contraintes (table 3) donne un facteur de duplication calculé $k_{calc} = 3.67$ et un facteur de duplication pratique $k_{prat} = 3$. Les cpp associés sont : $cpp_{calc} = 11/3.67 = 3$ et $cpp_{prat} = 11/3 = 6.67$.

3.0.6 Implantation registre

Pour le code (table 4), le système de contraintes (table 10) donne un facteur de duplication $k_{calc} = 7.33$ et un facteur de duplication pratique $k_{prat} = 7$. Comme $p = 1$, les cpp associés sont $cpp_{calc} = 11/7.33 = 1.50$

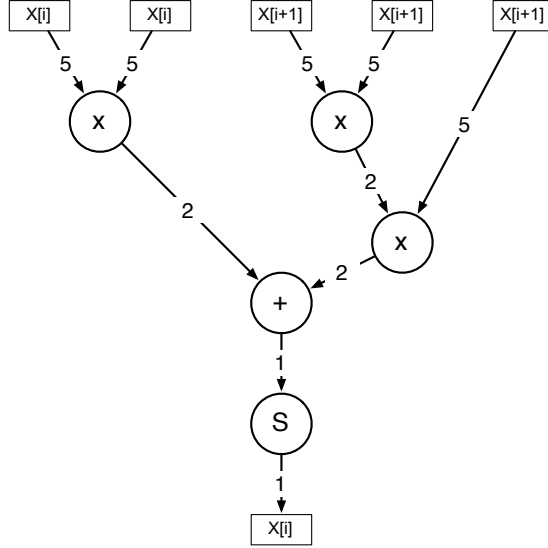


FIG. 2 – graphe de dépendance de donnée associé à l’implantation basique

type d’instructions	n_i	d_{imax}	c	d_i	k_i
I	10	8	11	$10/11 = 0.909$	8.8
E/S	6	2	11	$6/11 = 0.545$	3.67
M	3	2	11	$3/11 = 0.273$	7.33
A	1	4	11	$1/11 = 0.091$	44

TAB. 9 – système de contraintes *In Order* associé au code basique prenant en compte le plus long chemin

et $cpp_{prat} = 11/7 = 11.57$. Nota : s’il avait été possible d’avoir $k = 7.33$, nous aurions eu $cpp = 1.5$ qui correspond au premier système de contraintes.

3.0.7 Implantation avec Rotation de Registres

Pour le code (table 6), le système de contraintes (table 11) un facteur de duplication $k = 6$. et un nombre de cycles $cpp = 1.5$. Nota : cette fois, comme le facteur de duplication est un entier, le cpp calculé correspond à la valeur la plus basse possible trouvée avec le premier système de contraintes. Reste maintenant à trouver un ordonnancement en 1.5 cycles ...

4 Recherche de l’ordonnancement optimal

Les deux systèmes de contraintes proposent une solution en 1.5 cycles. Comme ce nombre de cycle n’est pas un entier, il faut rechercher le plus petit multiple de 1.5 qui soit un entier. Ici c’est 3 avec un facteur initial de duplication de 2. Comme on lance *en moyenne* un calcul tous les 1.5 cycles, lorsqu’on en lance deux en parallèle, on va lancer des calculs tous les 3 cycles. En prenant le chronogramme précédent on obtient le déroulage de la figure 5, qui pose un problème car il y a 4 multiplications au cycle 5. Il faut trouver un chronogramme qui soit duplicable.

Le chronogramme de la figure 6 est duplicable et ne pose pas de problème.

Il reste maintenant à identifier le prologue, le corps de la nouvelle boucle et l’épilogue. Pour identifier le corps de boucle, il faut rechercher le plus petit motif d’instruction qui se répète. Comme le chronogramme initial a été dupliqué d’un facteur 2 et que le cpp d’une itération est 1.5, il faut rechercher une suite d’instructions de longueur 3. Ces différentes portions de code sont identifiées figure 7. Le motif est bien de longueur 3, et on peut vérifier qu’il contient bien toutes les instructions de deux corps de boucle.

0	LL	$x_0=X[j]; x_1=X[i-1]$
1		
2		
3		
4		
5	xx	$t_0=x_0*x_0; t_1=x_1*x_1$
6		$X[i+1] * X[i+1]$
7	x	$t_1=t_1*x_1$
8		
9	+	$x_0*x_0 + x_1*x_1*x_1$
10	S	

FIG. 3 – chronogrammes associés à l’implantation registre

type d’instructions	n_i	d_{imax}	c	d_i	k_i
I	7	8	11	$7/11 = 0.636$	12.57
E/S	3	2	11	$3/11 = 0.273$	7.33
M	3	2	11	$3/11 = 0.273$	7.33
A	1	4	11	$1/11 = 0.091$	44

TAB. 10 – système de contraintes *In Order* associé au code Registre

Ainsi dans ce cas, les deux systèmes proposent une solution avec un $cpp = 1.5$ (ce qui n’est pas toujours le cas) et surtout, il existe un ordonnancement, ce qui n’est pas toujours le cas non plus.

0	Lx	$x_0=X[j]; t_1=x_1*x_1$
1		
2	x	$t_1=x_1*x_1*x_1$
3		
4		
5	x	$t_0=x_0*x_0$
6	=	$x_1=x_0$
7	+	$x_0*x_0 + x_1*x_1*x_1$
8	S	

FIG. 4 – chronogrammes associés à l’implantation Rotation de Registres

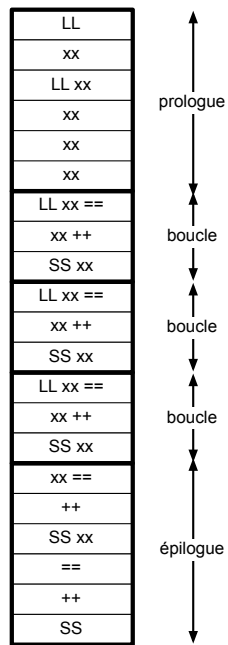


FIG. 7 – software pipelining du chronogramme