Calcul en virgule fixe

Lionel Lacassagne

LIP6
Université Pierre et Marie Curie (Paris 6)

Plan

- Présentation des formats de calcul en virgule fixe
- exemples de problèmes récurrents et de leur solutions

Calcul en virgule fixe

Pourquoi faire du calcul en virgule fixe ?

- parce que le calcul en virgule flottante n'est pas possible : pas de FPU !
  - FPGA trop petit pour instancier un processeur complet avec FPU (plus vrai de nos jours)
  - pas de FPU sur processeur / micro-controleur / DSP
- parce que le calcul en virgule flottante *coûte trop cher* !
  - prend trop de place sur FPGA
  - prend trop de temps sur processeur (latence des instructions)
- parce que le calcul en virgule fixe suffit
  - comprendre lorsque cela est suffisant
  - savoir calculer en virgule fixe

Cela est (continuellement) en train de changer avec l'intégration VLSI et la techno de gravure, de plus en plus de choses sont intégrable sur un composant discret...

Calcul en virgule fixe : quelques exemples

- DSP VLIW TMS 320C62x de Texas Instrument $[1998 : 2004]$
  - pas de calcul flottant
  - pas même de division (ni de modulo) en entier

  ... alors que c'est l'un des DSP les plus utilisés au monde (station de base GSM)

- Compression JPEG (la norme d'image la plus utilisée au monde)
  - en virgule fixe, car lors de la normalisation des calculs,
  - les calculs flottants étaient trop lents
  - Des tests ont montrés que pour une même taille d'image, si les calculs étaient fait en virgule flottante, cela améliorerait sensiblement la qualité des images ...

- Les algorithmes de décompression vidéo $\rightarrow$ VLC pour fournir un flux vidéo à 25 i/s en full-HD (de vidéos achetées légalement)

- Les algorithmes de compression vidéo pour ASIC (*Application Specific Integrated Circuit*) dans les *set-top boxes*, les téléphones portables et les caméras tout-terrains

Format *fixed-point Q*

$x = m.f$

- m : magnitude ou partie entière
- f : partie fractionnaire

Utilisation des nombres entiers codés en complément à 2

- max : $2^{m-1} - \frac{1}{2^f}$
- min : -2^{m-1}
- précision : $\varepsilon = \frac{1}{2^f}$

Exemple : $Q_{12.4}$

- nombres signés : $x \in [-2048..2047.9375]$
- nombres non signés : $x \in [0..4095.9375]$
- précision : $\varepsilon = 0.0625$

- Si un seul nombre est précisé, c'est la partie fractionnaire : Q_6, Q_8, Q_{10}
- souvent Q_8 ou Q_{16} sur processeur, mais tout est possible (FPGA, ASIC)
- et souvent les DSP ont des accumulateurs spéciaux sur plus de 32 bits (typiquement 40)

Le problème de la division

Il y a deux types d'architecture :

- celles où l'instruction de division entière présente : pas de problème
- celles où l'instruction de division entière absente : problème

Soit la division $y = x/d$

- si le diviseur d est une puissance de 2 ($d = 2^p$) : alors $y = x/2^p$ devient `y = x >> p`
- sinon utilisation d'une fraction équivalente en puissance de 2 : $1/d = \lambda/2^p$
- d'où : $\lambda = 2^p/d$. Ce calcul est fait une fois pour toute (tabulation ?), l'utilisateur choisi p en fonction de la précision nécessaire.
- Souvent $p = 8$: calcul en virgule fixe Q_8.

Le problème de la division - exemples

Exemples :

- moyenne de quatre points : $y(n) = \frac{x(n)+x(n-1)+x(n-2)+x(n-3)}{4}$
  `y[n]=(x[n]+x[n-1]+x[n-2]+x[n-3])»2`

- moyenne de trois points : $y(n) = \frac{x(n)+x(n-1)+x(n-2)}{3}$
  Q_8 : $\frac{2^8}{3} = 85.33 \Rightarrow \lambda = 85$
  `y[n]=(85*(x[n]+x[n-1]+x[n-2]))»8`
  Ne pas oublier les parenthèses !
  L'expression reste incomplète ...

Problème des fractions équivalentes

Comme d n'est pas une puissance de 2, $k \times d \neq 2^p$
Phénomène d'atténuation : $k \times d < 2^p$ (ici $3 \times 85 = 255$)

- peut être acceptable si algorithme non itératif,
- doit être pris en compte si algorithme itératif car risque de pertes importantes
  ex : $(\frac{255}{256})^{10} = 0.962$ perte de -3.8%, $(\frac{255}{256})^{100} = 0.676$ perte de -32.4%

Solution : remplacer le mode de calcul par troncature, par arrondi

- analogie calcul flottant : $round(1.4) = 1$, $round(1.6) = 2$
  $round(x) = \lfloor x + 0.5 \rfloor$
- mode troncature : $\frac{x}{d} = \lfloor \frac{x}{d} \rfloor$
- mode arrondi : $\frac{x}{d} = \lfloor \frac{x+d/2}{d} \rfloor$ la moitié du diviseur
- $\Rightarrow$ mode arrondi en Q_p : $\frac{x}{d} = \lfloor \frac{\lambda \times x + 2^{p-1}}{2^p} \rfloor$
- ex : Q_8 : `y[n]=(85*(x[n]+x[n-1]+x[n-2])+128)»8`

mais ce n'est pas tout ...

Précision et ordre des calculs intermédiaires

En mathématique (dans $\mathbb{R}$) les trois expressions sont équivalentes :

$$\frac{x(n)+x(n-1)+x(n-2)}{d} = \frac{x(n)}{d} + \frac{x(n-1)}{d} + \frac{x(n-2)}{d} = \frac{1}{d}x(n) + \frac{1}{d}x(n-1) + \frac{1}{d}x(n-2)$$

Mais en entier, deux phénomènes de perte de précision ont lieu :

1. Comme $\frac{1}{d} < 1$ et que le calcul est dans $\mathbb{N}$:
   - `(1/d)` est évalué avant la multiplication et vaut 0
   - l'expression `(1/d)*x0 + (1/d)*x1 + (1/d)*x2` est équivalente à
   - `y = 0*x0 + 0*x1 + 0*x2` soit zéro

2. Comme la division a lieu avant la sommation
   - il y a perte de la partie fractionnaire
   - l'expression `x0/d + x1/d + x2/d` est moins précise que `(x0 + x1 + x2) / d`

$\Rightarrow$ en général, on repousse *au plus tard* les opérations générant de la perte de précision, tout en faisant attention que les accumulations ne provoque pas d'*overflow*.

Précision des calculs intermédiaires

Exemple : soit la moyenne de trois valeurs successives $\frac{100+101+102}{3} = 101$

- $\lfloor \frac{100}{3} \rfloor + \lfloor \frac{101}{3} \rfloor + \lfloor \frac{102}{3} \rfloor = \lfloor 33.33 \rfloor + \lfloor 33.67 \rfloor + \lfloor 34.00 \rfloor = 100$
- $\lfloor \frac{85 \times 100}{256} \rfloor + \lfloor \frac{85 \times 101}{256} \rfloor + \lfloor \frac{85 \times 102}{256} \rfloor = \lfloor 33.20 \rfloor + \lfloor 33.54 \rfloor + \lfloor 33.87 \rfloor = 99$
- $\lfloor \frac{85 \times (100+101+102)}{256} \rfloor = \lfloor 100.61 \rfloor = 100$
- $\lfloor \frac{85 \times (100+101+102)+128}{256} \rfloor = \lfloor 101.11 \rfloor = 101$

Rappel : la moyenne d'un nombre impair de valeurs successives est la valeur du milieu

Dynamique des calculs

- il faut maximiser la dynamique des calculs intermédiaires pour repousser le plus tard possible les divisions
- mais il faut éviter les débordements de capacité (*overflow*)

Exemple du filtre binomial (coefficients du triangle de Pascal) appelé par abus de langage filtre "gaussien"

- $B_1 = \frac{1}{2}[\ 1 \quad 1\ ]$
- $B_2 = \frac{1}{4}[\ 1 \quad 2 \quad 1\ ]$
- $B_4 = \frac{1}{16}[\ 1 \quad 4 \quad 6 \quad 4 \quad 1\ ]$
- $B_8 = \frac{1}{256}[\ 1 \quad 8 \quad 28 \quad 56 \quad 70 \quad 56 \quad 28 \quad 8 \quad 1\ ]$

- La somme des coefficients est égale au dénominateur, donc
- si input sur 8 bits, la dynamique calculs intermédiaires pour B_8 est 2^{16} : OK en 1D jusqu'à 8.
- mais en 2D (voir cours sur filtrage) overflow des B_5...

Gestion des coefficients des filtres en TNS

filtre FIR : $y(n) = \sum_{k=0}^{m} h(k) \times x(n-k)$
On a $\sum_{k=0}^{m} h(k) = 1$
Donc en virgule fixe avec $H(k)$ valeur du coef $h(k)$ en arithmétique Q_p :
$H(k) = 2^p \times h(k)$
il faut **vérifiez que** $\sum_{k=0}^{m} H(k) = 2^p$

Exemple : lisseur récursif (IIR) de FGL (la sortie du filtre devient une entrée)

- $y(n) = b_0 x(n) + a_1 y(n-1) + a_2 y(n-2)$
- avec $b_0 = (1-\gamma)^2$, $a_1 = 2\gamma$, $a_2 = -\gamma^2$, $\gamma = e^{-\alpha}$
- paramètre du filtre : $\alpha \in [0.5, 1] \Leftrightarrow$ [lissage fort, lissage faible]
- si $\alpha = 0.5$ $b_0 = 0.155 \quad a_1 = 1.213 \quad a_2 = -0.368$
- Alors $B_0 = 39.68 \quad A_1 = 310.53 \quad A_2 = -94.21$
- par défaut la conversion de type en C réalise une troncature :
- `B0=39`, `A1=310`, `A2=-94` et `B0+A1+A2=255`. Au bout de 100 points, atténuation de 50%

Gestion des coefficients des filtres et quantification

Problème :

Hypothèse : à partir d'un certain rang le signal devient nul.

- $n < N$, $x(n) \neq 0$, y s'écrit $y(n) = b_0 x(n) + a_1 y(n-1) + a_2 y(n-2)$
- $n = N$, $x(n) = 0$, y devient $y(n) = a_1 y(n-1) + a_2 y(n-2)$
- comme $a_1 + a_2 < 1$, il y a une décroissance normale des valeurs du filtre.

Résultat : à partir d'un rang n' la sortie est nulle

- au rang n' : $y(n') = a_1 y(n'-1) + a_2 y(n'-2) = 0$
- mais au rang $n'+1$: $y(n'+1) = a_1 y(n') + a_2 y(n'-1) = a_2 y(n'-1)$
- or $a_2 < 0$ et $y(n'-1) > 0$ donc $y(n+1') < 0$
- en virgule fixe, un *overflow* apparait

Gestion des coefficients des filtres et quantification

Mais il y a pire ...

Exemple : lisseur de FGL avec $\alpha = 1$

- b_0=0.3996, a_1=0.7358, a_2=-0.1353 $\Rightarrow$ B_0=102.29, A_1=188.35, A_2=-34.65
- en entier : `B0`=102, `A1`=188, `A2`=-34, on a bien `B0+A1+A2`=256
- signal en entrée à 100 puis passe à 0

Résultat : *overflow* cyclique

n	0	1	2	3	4	5	6	7
x(n)	100	0	0	0	0	0	0	0
y(n)	100	60	30	14	6	2	0	255

n	8	9	10	11	12	13	14	15	16
x(n)	0	0	0	0	0	0	0	0	0
y(n)	187	103	50	23	10	4	1	0	255

Fonctions mathématiques manquantes sur certains processeurs / DSP

Jusqu'à maintenant la valeur d de la division était connue à l'avance. Possibilité de la remplacer par fraction équivalente.
Comment faire si ce n'est pas le cas ?

- utilisation d'une LUT (*Look-Up Table*)
- interpolation par morceau
- ré-écriture de la fonction

Utilisation d'une LUT

- la case d de la lut L contient la fraction équivalente à $\frac{1}{d} \simeq \frac{\lambda}{2^p}$:$L[d] = \lambda$
- la division $y = \frac{x}{d}$ est remplacée par une lecture : $y = \frac{L[d] \times x + 2^{p-1}}{2^p}$
- `y[n]=(L[d]*x + Q/2)»p`
- utilisation classique : tabulation des fonctions trigonométriques, ex $cos(\theta)$ par pas de 1 degré en flottant ou en virgule fixe

Problème : taille de la LUT : si les nombres sont sur 16 bits, il y a 2^{16} valeurs.

Fonctions mathématiques manquantes

Amélioration de la précision

- LUT avec meilleure précision (changement du pas)
- interpolation par morceaux : la valeur de la fonction est interpolée à partir des valeurs dans la LUT.
  plusieurs possibilité : interpolation linéaire (règle de trois) ou polynôme d'interpolation (interpolation cubique, spline, ...)
- polynôme d'interpolation (le polynôme doit être plus précis que pour l'interpolation par morceau).
- ré écriture de la fonction manquante

Fonctions mathématiques manquantes

Ré-écriture de la fonction manquante avec l'algorithme de Newton

- tangente en un point x_0 : $f(x) = (f(x_0) + (x - x_0)f'(x_0)$
- intersection avec l'axe x : $f(x) = 0$ soit $x = x_0 - \frac{f(x_0)}{f'(x_0)}$
- schéma itératif de Newton : $x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)}$

- on cherche à calculer $x_\infty = \frac{1}{d}$
- fonction réciproque : $f(x) = \frac{1}{x} - d$, $f'(x) = -\frac{1}{x^2}$
- schéma itératif : $\boxed{x_{n+1} = 2x_n - dx^2}$
- convergence quadratique : le nombre de bits corrects double à chaque itération.
- utilisation d'une petite LUT pour initialiser le calcul avec quelques bits corrects (par exemple $p/2$ bits, soit en Q_8 4 bits)

Fonctions mathématiques manquantes

Algorithme de Newton pour la virgule fixe

- schéma itératif flottant : $x_{n+1} = 2x_n - dx_n^2$
- soit B la base de calcul : $B = 2^p$
  (prendre $B = 10^p$ pour observer et reconnaitre la convergence).
- schéma itératif en virgule fixe : $\boxed{x_{n+1} = \dfrac{2Bx_n - dx_n^2 + B/2}{B}}$
- choix de la valeur initiale : faire attention aux problèmes de (non)convergence

Exemple : $d = 3$, $B = 100$, calculs en virgule flottante et virgule fixe

n	0	1	2	3	4	5	6	7	8	9
x_n	1	1.97	3.82	7.21	12.86	20.76	28.59	32.66	33.32	33.33
x_n, B = 100	1	2	4	8	14	22	29	33	33	33
x_n, B = 256	1	2	4	8	15	27	45	66	81	85

On a bien $33 = (100 \times 1)/3$ et $85 = (256 \times 1)/3$ le numérateur de la fraction équivalente
Attention, ne pas interpréter la ligne du bas comme la version arrondie de la ligne du dessus : 14 n'est pas l'arrondi de 12.86 ...

Fonctions mathématiques manquantes

Algorithme de Newton pour la virgule fixe

Mais comment faire si $d < 1$ car pas représentable en entier ?

- multiplier par B le schéma itératif
- soit $D = 2^p \times d$
- second schéma itératif entier : $\boxed{x_{n+1} = \dfrac{2B^2 x_n - D x_n^2 + B^2/2}{B^2}}$
- Attention à la dynamique des calculs (compromis dans choix de B)

Exemple : $d = 0.25, B = 100 \Rightarrow D = 25$
(au moins 1 calcul en flottant : $D = B \times d$)

n	0	1	2	3	4	5	6
x_n	1	2.00	3.99	7.93	15.70	30.79	59.21
x_n	1	2	4	8	16	31	60

n	7	8	9	10	11	12	13
x_n	109.66	189.25	288.96	369.18	397.62	399.99	**400.00**
x_n	111	191	291	370	398	**400**	**400**

Fonctions mathématiques manquantes

Inverse de la fraction équivalente : λ^{-1}

Troisième schéma itératif : $\boxed{x_{n+1} = \dfrac{2B^2 x_n - \lambda x_n^2 + B^2/2}{B^2}}$

Exemple : $\lambda = 33$ (résultat du premier schéma itératif), $B = 100$

n	0	1	2	3	4	5	6
x_n	1.00	2.00	3.98	7.91	15.61	30.42	57.78
x_n	1	2	4	8	16	31	59

n	7	8	9	10	11	12	13
x_n	104.54	173.02	247.25	292.76	302.68	**303.03**	**303.03**
x_n	106	175	249	293	302	**303**	**303**

Erreur de convergence normale (liée à l'approximation de $1/3 \simeq \lfloor 100/3 \rfloor$), mais on a bien $d = \lambda^{-1} = x_n/B = 3$

Equation aux dimensions

Ne pas faire n'importe quoi en arithmétique : ramener les fractions au même dénominateur

Soient $a, b \in \mathbb{F}$ et $A, B \in \mathbb{Q}$ leurs équivalents quantifiés sur q bits.

On souhaite que le résultat soit avec la même quantification

- $\boxed{a + b \rightarrow A + B}$ car $A/Q + B/Q = (A + B)/Q$ (même base), mais
- $\boxed{a + 1 \rightarrow A + Q}$ car $A/Q + 1/1 = (A + Q)/Q$
- $\boxed{a \times b \rightarrow A \times B/Q}$ car $A/Q \rightarrow B/Q = (A + B)/Q^2$

Si on souhaite changer la quantification du résultat

Si $A_1 \in \mathbb{Q}_1, A_2 \in \mathbb{Q}_2$ (hypothèse $\mathbb{Q}_1 < \mathbb{Q}_2$) : $\Delta_Q = Q_2/Q_1$, $\delta_q = q_2 - q_1$

- $a_1 + b_2 \rightarrow \Delta_Q \times A + B \in \mathbb{Q}_2$ ou
- $a_1 + b_2 \rightarrow A + B/\Delta_Q \in \mathbb{Q}_1$
- ou tout autre valeur à l'intérieur de $[Q_1, Q_2]$: à l'utilisateur de choisir
- ou tout autre valeur à l'extérieur de l'intervalle : au delà = aucune utilité, en de ça = pour des raisons d'optimisation ou de contrainte mémoire (seuls les nombres sur 32, 16 ou 8 bits existent ... sur CPU, mais sur FPGA, plus de liberté).

Quelque(s) remarque(s) supplémentaire(s) #1

Bases de calcul et quantifications binaire, décimale ou hexa

Soient 4 nombres $a, b, c, d \in Q$ on sait que :

- l'addition de 2 nombres sur Q bits donne un nombre sur $Q+1$ à cause de la retenue possible : $9+9=18$ (ici $1+1=2$ digits)
  $t_1 = a+b, t_2 = c+d, \Rightarrow t_1 \in (Q+1), t_2 \in (Q+1)$
- l'addition de 2 nombres sur $(Q+1)$ bits donne un nombre sur $Q+2$ à cause de la retenue possible : $99+99=198$ (ici $2+2=3$) digits)
  $t_= t_1 + t_2, \Rightarrow t \in (Q+2)$
- mais l'addition de 4 nombres sur Q bits ne donne nécessairement un nombre sur $Q+2$: $9+9+9+9=36$ (ici $1+1+1+1=2$ digits)
- le raisonnement est faux : mélange entre base 2 et base 10 :
  $(1_2+1_2)+(1_2+1_2) = 10_2+10_2 = 100_2$
  $(7_8+7_8)+(7_8+7_8) = 16_8+16_8 = 34_8$
  $(9_{10}+9_{10})+(9_{10}+9_{10}) = 18_{10}+18_{10} = 36_{10}$
  $(F_{16}+F_{16})+(F_{16}+F_{16}) = 1E_{16}+1E_{16} = 3C_{16}$
- lien entre addition et multiplication : il faut $Q+2$ additions en base Q pour provoquer plus qu'un bit/digit de retenue

Quelque(s) remarque(s) supplémentaire(s) #2

Q additions sans retenue supplémentaire

Quelques exemples :

- Base 2 : $1_2 + 1_2 + 1_2 = 3 \times 1 = 3 = (11)_2$ et $4 \times 1 = 4 = (100)_2$
- Base 8 : $9 \times 7 = 63 = (77)_8$ et $10 \times 7 = 70 = (106)_8$
- Base 10 : $11 \times 9 = (99)_{10}$ et $12 \times 9 = (108)_{10}$
- Base 16 : $17 \times 15 = 255 = (FF)_{16}$ et $18 \times 15 = 270 = (10E)_{16}$

La preuve :
Q additions de la valeur max $(Q-1)$ donne $(Q+1) \times (Q-1) = Q^2 - 1 < Q^2$
et en plus c'est optimal : la somme est à une unité de la retenue.

$\Rightarrow$ il faut faire attention au regroupement des additions pour borner au mieux les sommes...

Quelque(s) remarque(s) supplémentaire(s) #3

Et si les nombres sont négatifs ou qu'il y a des soustractions, alors cela se complique ... c'est le domaine du calcul par intervalle.

Conclusion

- Le calcul en virgule fixe est rapide mais contient de nombreux pièges,
- facilement identifiables et évitables dès lors qu'on sait qu'ils existent...
- Les fonctions manquantes sont facilement re-créables à partir de l'arithmétique existante
- Newton est votre meilleur ami ...