Opérateurs arithmétiques matériels
Diviseurs

Florent de Dinechin

7 décembre 2005

Intro

Division ou racine carrée, c'est kif-kif

L'inverse (resp. la racine carrée) de X c'est le nombre que, quand on le multiplie par X (resp. par lui-même) on obtient 1 (resp. X).
Les algos étant dérivés de cette formulation, on aura des choses qui ressemblent pour les deux opérations.
Deux grandes classes d'algos :

- Récurrence de chiffres (division comme à la main)
  - Opération de base : addition
  - Convergence : linéaire
- Itérations à la Newton
  - Opération de base : multiplication
  - Convergence : quadratique

Faut-il du matériel pour division et racine carrée ?

Il y a des papiers de la bande à Flynn à ce sujet.

- En gros, une opération arithmétique sur 20 est une division
- Si la division est 20 fois plus lente que les autres, elle prendra autant de temps.

(la valeur de 20 varie selon les domaines d'application considérés.)

Notations

- On divise X (dividende) par D (diviseur) pour obtenir un quotient Q et un reste R
- le tout vérifiant
$$X = QD + R$$
- Conditions pour l'unicité de (Q, R) :
  - $|R| < D.\mathrm{ulp}(Q)$
  - et R a le signe de X
- On ne va pas s'embêter avec des nombres négatifs
  - rappelez moi de vous persuader au fur et à mesure que ce n'est pas intéressant
- Comme d'hab, je me débarasse de la virgule
  - Intuitivement comme à la main,
  - proprement en mettant des puissances de β où il faut.
  - Diviseurs utiles : entiers et parties fractionnaires

Humilions un étudiant de M2 IF

... en l'envoyant au tableau poser la division de 335 par 21

L'algo à la main

- On calcule des tas de restes partiels, qu'on notera toujours R_j avec
  - $R_0 = X$
  - $R = R_{\text{final}}$
- L'itération consiste à
  - intuiter un chiffre q_i du quotient
    - Difficile parce que cela revient à faire une division approchée dans sa tête
    - D'ailleurs, en primaire, on se plantait parfois.
  - calculer le nouveau reste partiel comme $R_{i+1} \approx R_i - q_i D$ (j'ai écrit $\approx$ pasque il y a un décalage caché dedans)

Écrit proprement

- Plaçons nous dans le cas entier
  - en base β pour tout le monde, chiffres dans $\{0...\beta-1\}$
  - avec X sur $2n$ chiffres, D sur n chiffres, et Q qui ne déborde pas
  - "Q qui ne déborde pas" stadire $X/D < \beta^n$, ou encore $X < \beta^n D$
  - posons (dans un coin) $D' = \beta^n D$, on n'a pas fini de le voir
- L'algo qui gère proprement le décalage dans ce cas est :

1: $R_0 = X$
2: **for** $j \in \{0..n-1\}$ **do**
3: Calculer $R_{j+1} = \beta R_j - q_{n-j-1} D'$
4: en ayant intuité un q_{n-j-1} tel que $0 \leq R_{j+1} < D'$
5: **end for**

- La condition sur le chiffre intuité, c'est celle qu'on cherche à vérifier à la main

Écrit proprement

1: $R_0 = X$
2: **for** $j \in \{0..n-1\}$ **do**
3: Calculer $R_{j+1} = \beta R_j - q_{n-j-1} D'$
4: en ayant intuité un q_{n-j-1} tel que $0 \leq R_{j+1} < D'$
5: **end for**

- La condition sur le chiffre intuité, c'est celle qu'on cherche à vérifier à la main
  - Si q_{n-j-1} est trop petit on aura R_{j+1} négatif
  - S'il est trop petit on aura $R_{j+1} \geq D'$
  - Ce sera fâcheux à l'itération suivante : plus aucun $q_{n-j} \in \{0...\beta-1\}$ ne pourra faire l'affaire
  - (quand on sort du domaine de convergence on n'y rentre plus, et donc le reste final est soit trop grand soit négatif)
- A-t-on le choix du q_{n-j-1} ?

Humilions un autre étudiant d'excellence

... en l'envoyant au tableau poser en binaire la division de 25 par 5 avec $n = 3$ bits.

Si tout va bien...

... la victime précédente a intuité l'algorithme dit de division restaurante.

- En binaire c'est facile car on n'a que deux choix, donc on en essaye un.
- mais faut plein d'itérations
- Avec D en base β plus grande,
  - on aura moins d'itérations
  - mais si l'ensemble des chiffres est non redondant il n'y a qu'un q_j possible à chaque itération
  - et la fonction qui le choisit dépend de tout R_j et de tout D

Le rapport avec les restaurants ?

Pourquoi la division restaurante s'appelle comme cela ?

1: $R_0 = X$
2: **for** $j \in \{0..n-1\}$ **do**
3: Calculer $2R_j - D'$
4: Si c'est positif ou nul, $q_{n-j-1} = 1$ et $R_{j+1} = 2R_j - D'$
5: sinon, $q_{n-j-1} = 0$ et $R_{j+1} = 2R_j$
6: **end for**

Sur un vieux processeur, on ne veut pas utiliser plus de registres que les registres contenant initialement X et D. On implémentera donc comme cela :

1: $R_0 = X$
2: **for** $j \in \{0..n-1\}$ **do**
3: Calculer $R_{j+1} = 2R_j - D'$ tout dans le même registre
4: Si R_{j+1} positif ou nul, $q_{n-j-1} = 1$
5: sinon, $q_{n-j-1} = 0$ et $R_{j+1} += D'$ on a perdu $2R_j$
6: **end for**

Un ptit changement de variable

- On pose $\tilde{R}_j = 2R_j - D'$
- Avantage : la division restaurante calculait
  - soit $R_j - D'$
  - soit R_j par $R_j - D' + D'$ (et cela prenait deux cycles)
- Maintenant on devra calculer
  - soit $\tilde{R}_j - D' = 2R_j - 2D' = 2R_{j+1}$
  - soit $\tilde{R}_j + D' = 2R_j = 2R_{j+1}$
- Avantage : une seule addition par cycle, jamais deux (et pas besoin de plus de registres que la division restaurante)
- Inconvénient : $\tilde{R}_j$ est signé (mais on a $|\tilde{R}_j| < D$)

Division non restaurante

- R_j est signé et $|R_j| < D$
- Algo :

1: $R_0 = X$
2: $R_1 = 2R_0 - D'$
3: **for** $j \in \{1..n-1\}$ **do**
4: **if** $R_j \geq 0$ **then**
5: $q_{n-j-1} = 1$ et $R_{j+1} = 2R_j - D'$
6: **else**
7: $q_{n-j-1} = 0$ et $R_{j+1} = 2R_j + D'$
8: **end if**
9: **end for**
10: Correction si le dernier reste est négatif :
11: **if** $R_n \geq 0$ **then**
12: $q_0 = 1$
13: **else**
14: $q_0 = 0$ et $R_n = R_n + D'$
15: **end if**

Remarques sur la division non restaurante

- On a gagné un cycle par itération, mais on a perdu une itération (la correction finale).
- Autre manière de voir : c'est le premier algo en prenant les q_i dans $\{-1, 1\}$ (sans 0)
- Certains contrôleurs comme le ST200 ont une instruction qui fait en un cycle une itération de division non restaurante (mais je sais pas vraiment pourquoi pas la restaurante).

Cela dit tout cela c'est pas brillant

Deux problèmes :

- Fonction de sélection (qui produit les q_j)
  - dépend de tous les chiffres de R_j et de D
  - compliquée en grande base (plus de deux choix possibles)
- La division se pipeline mal :
  - par exemple en binaire faut avoir calculé tous les bits pour décider le signe
  - donc notre truc habituel de calculer les R_j en carry-save ou autre système redondant ne marche pas
  - Donc on peut facilement construire un diviseur cellulaire rectangulaire, comme pour les multiplieurs, mais son délai sera de n^2 (pour le multiplieur, $2n$)

La solution va être de chercher les q_j dans un ensemble de chiffres redondants

Division additive (récurrence de chiffre)

Division fractionnaire pour les FPU

$$X = QD + R$$

- Celle qu'il faut pour un diviseur flottant c'est
  - Tout le monde sur n chiffres
  - $1/2 \leq X < 1$
  - $1/2 \leq D < 1$
  - ce qui donnera $1/2 < Q < 2$
  - donc faudra le normaliser
    (le ramener à $1/2 \leq Q < 1$ en décrémentant l'exposant).
- Pour simplifier les notations dans la suite on imposera juste $0 \leq Q < 1$
  - mais vu les domaines de X et D on n'aura jamais $Q < 1/2$
- X et D sont en base 2 au départ, on pourra les considérer en base $\beta = 2^k$ si cela nous arrange

Reprenons à zéro

- On veut obtenir Q et R tels que

$$\begin{cases} X = DQ + R \\ |R| < D.\text{ulp}(Q) \end{cases}$$

- Q va être calculé en base β (une petite puissance de 2) avec un ensemble de chiffres redondants :

$$q_j \in \{-\alpha ... \alpha\} \quad \text{avec} \quad \alpha \geq \lceil \beta/2 \rceil$$

- Du coup $\text{ulp}(Q) = \beta^{-n}$
- On va calculer les chiffres q_i de Q en séquence
- Soit $Q_j = \sum_{i=1}^{j} q_i \beta^{-i}$ (quotient partiel, $0 \leq Q_j < 1$)
- Ainsi on aura $Q = Q_n$ (à une petite correction finale près)
- Et donc on veut $0 \leq X/D - Q < \beta^{-n}$

Au boulot

- Posons $\epsilon_j = X/D - Q_j$
  - On veut au final $0 \leq \epsilon_n < \beta^{-n}$
  - Imposons cette condition (au signe près car nos chiffres sont signés) à chaque itération : $|\epsilon_j| < \beta^{-j}$
- En multipliant tout par D : $|X - DQ_j| < D\beta^{-j}$
- On pose naturellement $R_j = \beta^j(X - DQ_j)$ (reste partiel décalé)
- Et on veut donc $-D < R_j < D$
- En calculant R_{i+1} on retombe alors sur notre récurrence :

$R_0 = X$
for $j \in \{1..n\}$ **do**
  $q_{j+1} = \mathrm{Sel}(R_j, D)$
  $R_{j+1} = \beta R_j - q_{j+1} D$
end for

Petites misères des chiffres signés

- Plus une petite correction pour avoir notre reste final positif

if $R_n \geq 0$ **then**
  $Q = Q_n$
  $R = \beta^{-n} R_n$
else
  $Q = Q_n - \beta^{-n}$
  $R = \beta^{-n}(R_n + D)$
end if

Récapépetons

Le cœur de la récurrence c'est

1: $R_0 = X$
2: **for** $j \in \{1..n\}$ **do**
3: $\quad q_{j+1} = \mathsf{Sel}(R_j, D)$
4: $\quad R_{j+1} = \beta R_j - q_{j+1} D$
5: **end for**

Là dedans il y a trois calculs à faire :

- le choix du chiffre q_{j+1}
- la multiplication de ce chiffre par D
- la soustraction

Quels q_{j+1} peut-on choisir ?

1: $R_0 = X$
2: **for** $j \in \{1..n\}$ **do**
3: $\quad q_{j+1} = \mathrm{Sel}(R_j, D)$
4: $\quad R_{j+1} = \beta R_j - q_{j+1} D$
5: **end for**

La condition $-D < R_j < D$ est nécessaire mais pas suffisante pour assurer la convergence de l'algorithme. Intuitivement,

- maintenant notre ensemble de chiffres déborde de l'ensemble de chiffres minimal,
- donc il faut réduire en conséquence le domaine des R_j
- On va calculer $\underline{B}$ et $\overline{B}$ telles que
  $\underline{B} \leq R_j \leq \overline{B} \implies \underline{B} \leq R_{j+1} \leq \overline{B}$

Quels q_{j+1} peut-on choisir ?

- Supposons $\underline{B} \leq R_j \leq \overline{B}$
- On définit $[L_k U_k]$ l'intervalle de sélection pour le chiffre k comme le plus grand intervalle de valeurs de βR_j pour lequel qu'on peut choisir $q_{j+1} = k$ et avoir encore $\underline{B} \leq R_{j+1} \leq \overline{B}$ (cet intervalle sera indépendant de j)
- On veut $\underline{B} \leq R_{j+1} \leq \overline{B}$

$$R_{j+1} = \beta R_j - q_{j+1} D = \beta R_j - kD$$

lorsque $L_k \leq \beta R_j \leq U_k$
- d'où $L_k = \underline{B} + kD$ et $U_k = \overline{B} + kD$

Quels q_{j+1} peut-on choisir ?

- $L_k = \underline{B} + kD$ et $U_k = \overline{B} + kD$
- U_k croît avec k, donc pour le plus grand chiffre $k = \alpha$ ce sera la plus grande valeur de βR_j possible, soit $U_\alpha = \beta\overline{B}$.
- De même $L_{-\alpha} = \beta\underline{B}$
- En reportant dans les deux équations précédentes on obtient $\underline{B} = \beta\underline{B} + \alpha D$ et $\overline{B} = \beta\overline{B} - \alpha D$
- d'où

$$\underline{B} = -\frac{\alpha}{\beta - 1} D$$

et

$$\overline{B} = \frac{\alpha}{\beta - 1} D$$

- Reste plus qu'à poser $\rho = \dfrac{\alpha}{\beta - 1}$ (facteur de redondance)
  - comme $\alpha \geq \lceil \beta/2 \rceil$ on a $1/2 < \rho \leq 1$
  - Rem : pour $\rho = 1$ (ensemble de chiffres maximalement redondant) on n'assure plus $-D < R_n < D$ (strict), on pourra s'en sortir en bricolant la dernière itération

Récapépètons

Tout cela c'était pour construire la fonction Sel(R_j, D) :

- On a déterminé des intervalles de sélection $[L_k\, U_k]$ avec

$$\begin{cases} L_k = (-\rho + k)D \\ U_k = (\rho + k)D \end{cases} \qquad \rho = \frac{\alpha}{\beta - 1}$$

- Ce sont des intervalles des valeurs de βR_j, dépendant de D mais indépendants de j
- tels que à chaque j, si $\beta R_j \in [L_k, U_k]$, alors on peut choisir tranquillement $q_{j+1} = k$
- (autrement dit on peut choisir q_{j+1} parmi les k tels que $\beta R_j \in [L_k, U_k]$)
- et en respectant cette condition pour tout j on aura $\forall j\ R_j \in [-\rho D, \rho D]$

ce qui nous assure $R_n \in]-D, D[$ (ou presque dans le cas $\rho = 1$)

Bref

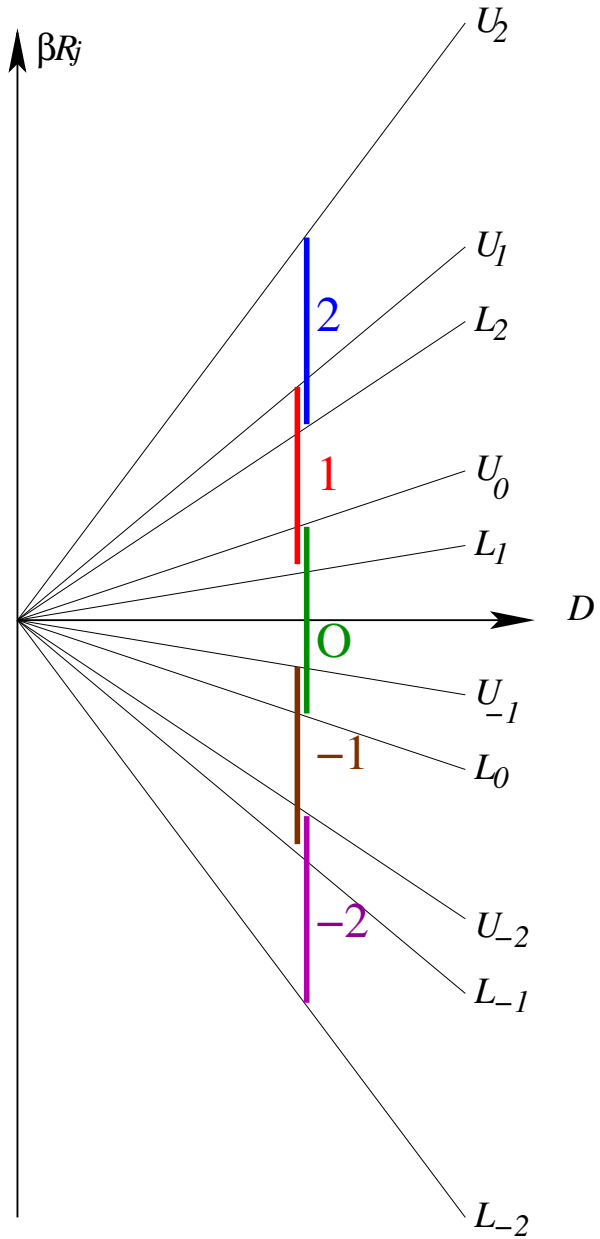
Bref on a formalisé la latitude de choix que permet la redondance pour Sel(R_j, D).
Dans la suite on va exploiter cette latitude pour faire des fonctions Sel() simples.

Diagramme de Robertson

- R_{j+1} en fonction de βR_j, dans un rectangle qui représente les bornes
- Une ligne par chiffre k, représentant la récurrence $R_{j+1} = \beta R_j - kD$
- Les $[L_k U_k]$ sont les projetés de ces lignes

On fait des lignes continues, mais en fait les axes sont discrets

Diagramme P-D



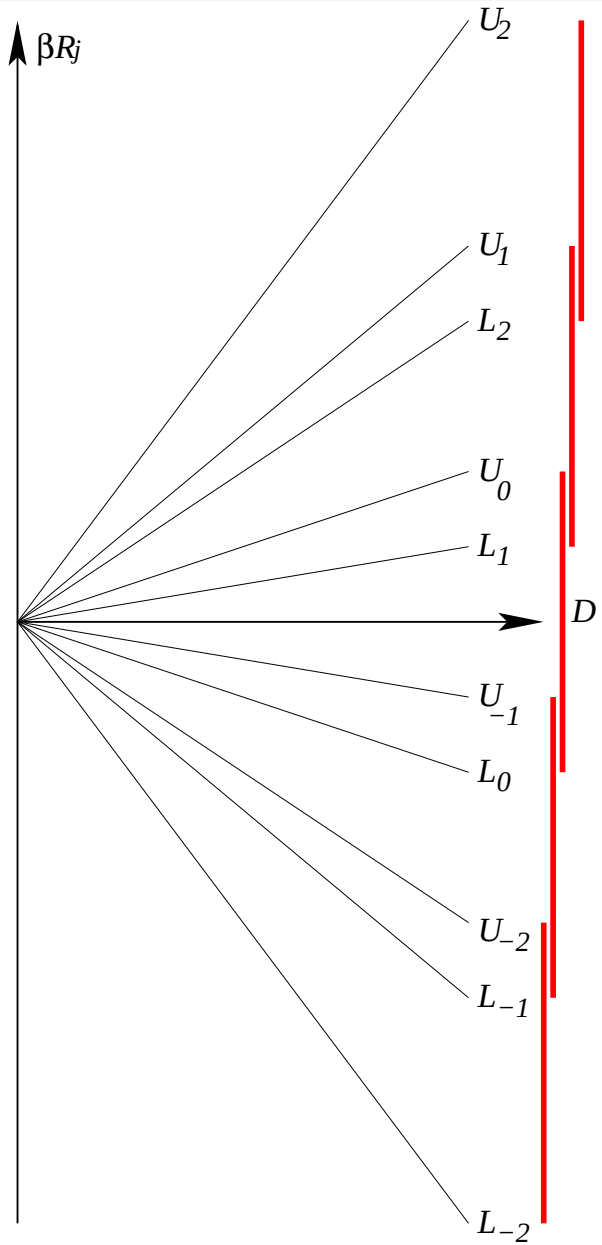
- Axes : le diviseur D et le résidu décalé βR_j
- Deux lignes par chiffre k, représentant L_k et U_k
- Pourquoi il s'appelle P-D ?
- La fonction Sel(R_j, D) découpe le diagramme P-D en zones sur lesquelles elle est constante
- Exemple : $\beta = 4$, $\alpha = 2$, $\rho = 2/3$ (bug du pentium)

Au fait, peut-on toujours sélectionner un chiffre ?

- Oui si chaque valeur possible de βR_j appartient au moins à un $[L_k U_k]$
- Ce qui s'écrit $L_k - \beta^{-n} \leq U_{k-1}$
- Prenons une condition plus forte : $L_k \leq U_{k-1}$
- et remplaçons par les valeurs calculées, on obtient ...
- $\rho \geq 1/2$, qui est toujours vrai.

Donc pas de problème.

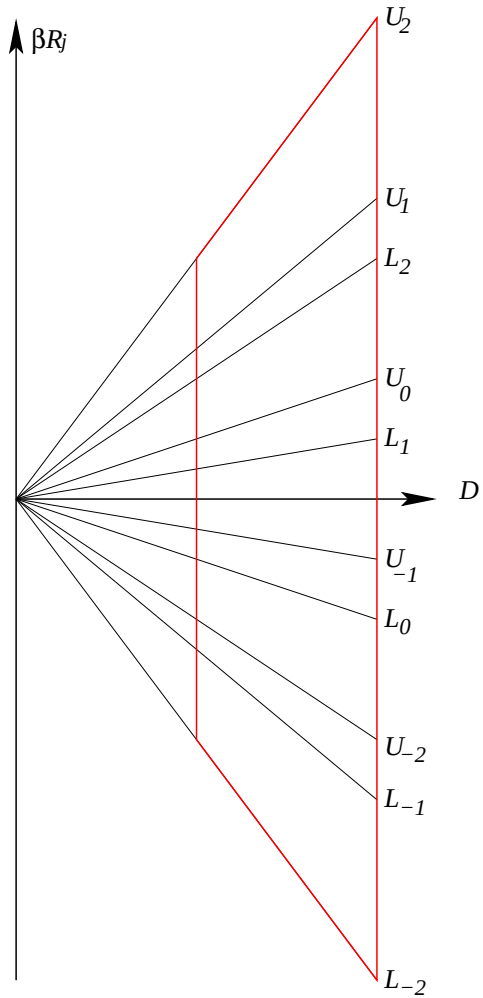
Recouvrement des $[L_k U_k]$ et latitude de choix



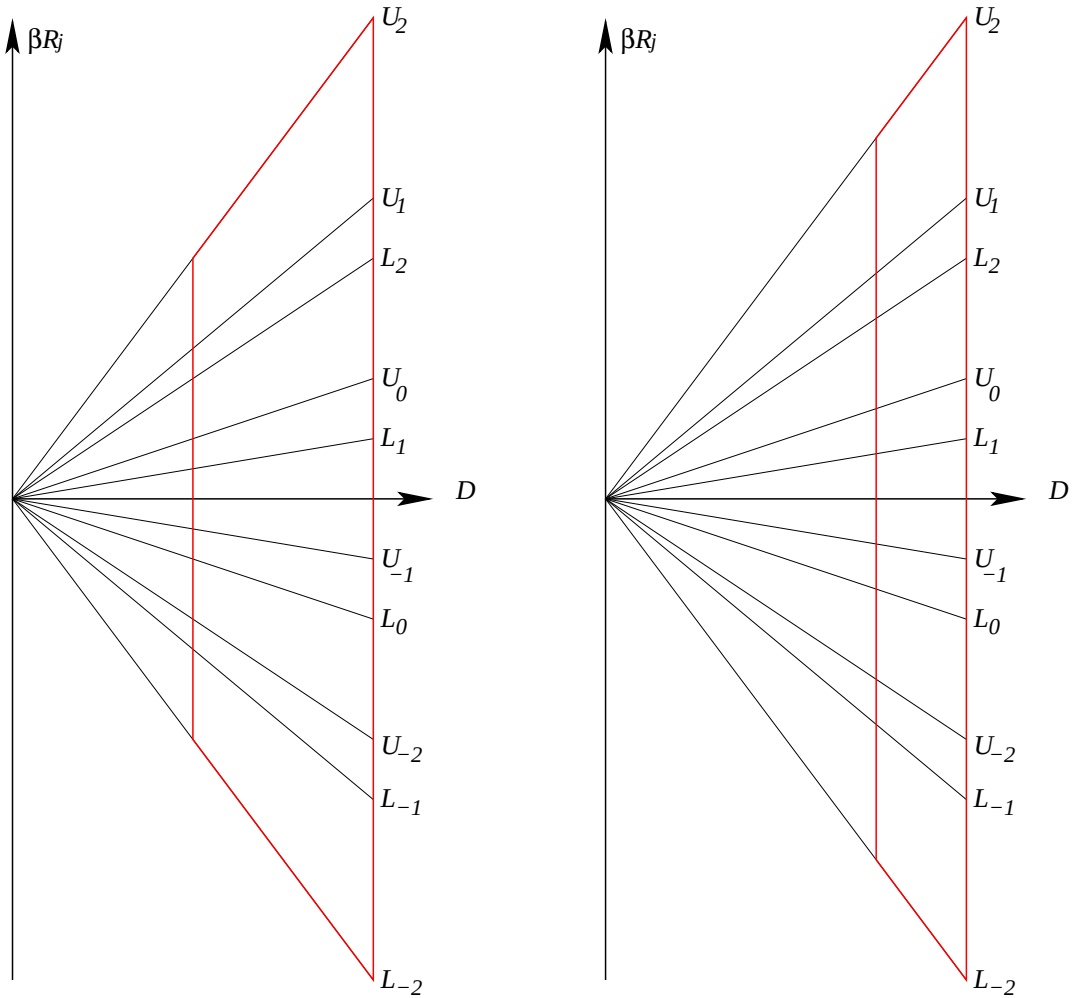
- Plus il y a de recouvrement, plus on a de choix
- Plus on aura le choix, moins on aura à regarder de bits de βR_j et de D pour faire un choix
- Taille d'un recouvrement : $U_{k-1} - L_k = (2\rho - 1)D$

Prescaling

- Le recouvrement dépend de D, donc on a moins de choix dans les petits D
- heureusement D est normalisé ($D \in [1/2, 1[$)
- et on pourra même le *préscaler* pour rapprocher D de 1, par exemple
  - si $d_2 = 1$, alors $D \in [\frac{3}{4}, 1[$
  - si $d_2 = 0$, alors $D \in [\frac{1}{2}, \frac{3}{4}[$: on multiplie X et D par $3/2$
  - Dans les deux cas, $D \in [\frac{3}{4}, \frac{9}{8}[$



Prescaling



En préscalant assez, Sel ne dépendra plus que de βR_j

Une première architecture (pas trop tôt)

- Construire des multiples du diviseur qui sont
  - faciles à calculer, et
  - dans la partie de recouvrement
- et les comparer à βR_j
- Autrement dit découper le diagramme P-D par des droites

Exemple classique avec $\beta = 4$, $\alpha = 2$:

- $\rho = 2/3$
- $I_2 = [4D/3, 8D/3]$, $I_1 = [D/3, 5D/3]$, $I_0 = [-2D/3, 2D/3]$ ...
- $I_1 \cap I_2 = [4D/3, 5D/3]$, prenons $P_{1-2} = 3D/2$
- $I_0 \cap I_1 = [D/3, 2D/3]$, prenons $P_{0-1} = D/2$
- ...

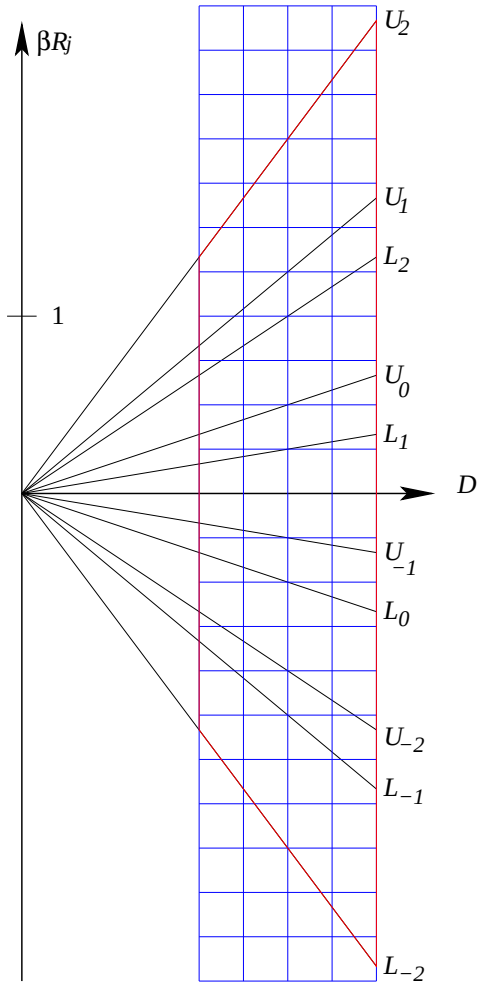
D'où l'algo :

- On précalcule en parallèle $3D/2$, $D/2$, $-D/2$, $-3D/2$
- à chaque itération
  - On compare en parallèle βR_j à toutes ces valeurs
  - Ceci détermine un unique chiffre q_{j+1}

Une autre idée

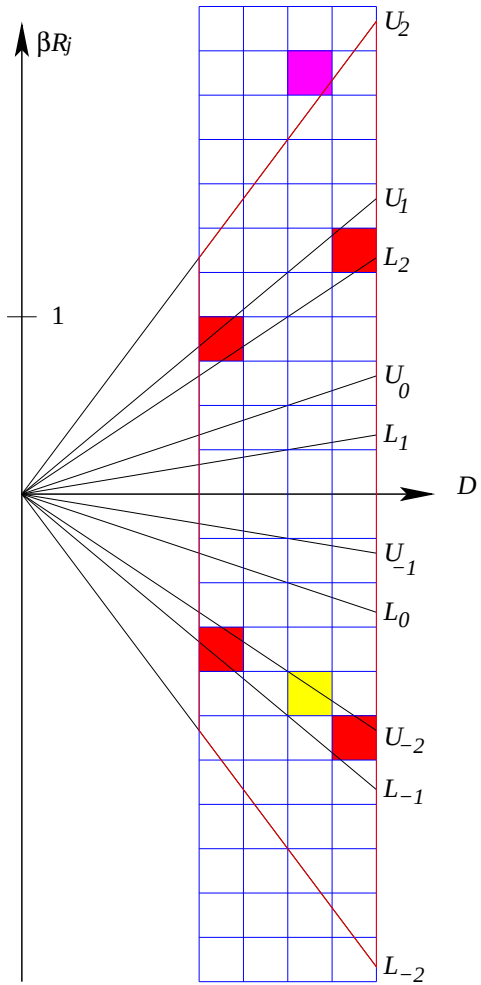
- idée : utiliser des valeurs de βR_j et D tronquées à quelques bits (le moins possible)
- $q_{j+1} = \mathrm{Sel}(\{\beta R_j\}_h, \{D\}_w)$
- graphiquement, paver le diagramme P-D par des blocs de taille 2^w par 2^h
- on a tout ce qu'il faut pour exprimer cela formellement

La figure c'est toujours $\beta = 4$, $\alpha = 2$, en ne regardant que 2 bits de D et 5 de βR_j.
Exercice de coloriage : remplissez les cases avec des chiffres, et peignez les cases pas remplissables en rouge.
Exercice d'archi première année : quelles cases adjacentes sont regroupables ?



Corrigé

La figure c'est toujours $\beta = 4$, $\alpha = 2$, en ne regardant que 2 bits de D et 5 de βR_j.
C'est presque bon, il faudra prendre 3 bits de D et 6 de βR_j
Le *prescaling* par 3/4 permettra de prendre (3,5) bits.
Attention, si βR_j est en complément à 2, le dessin discret n'est pas symétrique ! Les pavés incluent leur frontière inférieure mais pas leur frontière supérieure.
Exercice d'archi première année : quelles cases adjacentes sont regroupables ? Que fait-on des cases en dehors du trapèze utile ? Attention au bug du Pentium.
Ceux que cela amuse, ceci est mis en équations dans les bouquins d'Ercegovac et Lang.



C'est pas tout

Maintenant la sélection du chiffre est une petite table, mais qui dépend encore des poids *forts* de βR_j... ceux qui arrivent en dernier.
Pour aller plus vite, deux options

- calculer R_j sans propagation de retenue (dans un système redondant)
- "prédire" un q_{j+1} avant la fin du calcul de βR_j

Les restes partiels en redondant

- Rappel : $R_j = \beta R_{j-1} - q_j D$
  - Il faut déjà savoir calculer $q_j D$ sans propagation de retenue
  - exemple : base 4, chiffres $\{$-2...2$\}$ pour Q, carry-save pour R_j : facile.
  - autre exemple : base 4, chiffres $\{$-3..3$\}$ pour Q, carry-save pour R_j : quel matériel faut-il ?
  - et en base 8 ?
- Inconvénient : il faudra plus de bits de βR_j pour choisir q_{j+1} (redondance)

Prédiction de chiffre du quotient

"prédire" un q_{j+1} avant la fin du calcul de βR_j

- Rappel : $R_j = \beta R_{j-1} - q_j D$
- utiliser une approximation de R_j à partir des t bits de poids fort de R_{j-1}
- $R_j \approx R_j^T = \beta\{R_{j-1}\}_t - \{q_j D\}_t$
- ainsi on calcule en parallèle R_j et q_{j+1}

Le bug du Pentium

Comment perdre 475 000 000 de dollars ?

- Quotient en base 4, q_j dans $\{-2, -1, 0, 1, 2\}$
- βR_j en carry-save
- Conversion en binaire approchée (addition des 7 bits de poids forts seulement)
- Fonction de sélection : table utilisant ces 7 bits de poids forts et 5 bits de D
- 5 entrées erronées dans la table (sur la droite du haut du diagramme PD)

Retour sur le *prescaling*

- Dans le cas où le reste est en redondant, on a un multiadditionneur (m en 2) pour calculer notre reste partiel
- Essayons de l'utiliser pour le *prescaling*.
- Pour calculer $D' = SD$ on doit déterminer des facteurs S (en fait $S(D)$) qui se calculent en m additions
- Le mieux qu'on puisse faire c'est exprimer $S(D)$ en base 4 dans l'ensemble de chiffres $\{-1, 0, 1, 2\}$ (qui se font tous par décalage)
- soit $S = \sum_{i=0}^{m-1} s_i 4^{-i}$
- avec $s_0 \in \{1, 2\}$ pasque $1 \leq S \leq 2$
- Bref le multiadditionneur m en 2 permet de multiplier par $4^{m-1} + 1$ valeurs (consécutives) de S

Retour sur le *prescaling*

- Donc le multiadditionneur m en 2 permet de multiplier par $4^{m-1}+1$ valeurs (consécutives) de S
- On veut les déterminer par une table, fonction de s bits de poids fort de D
- ce qui nous coupe l'intervalle de D en 2^{s-1} sous-intervalles sur lesquels S sera constant
- d'où la condition $2^{s-1} \geq 4^{m-1}+1$ soit $s \geq 2m-1$

Retour sur le *prescaling*

- On pourra prendre $s = 2m - 1$ mais plus s est grand plus on peut réduire $\max_D(1 - D')$
- On peut aussi fixer des bornes $1 - \epsilon_- < D' < 1 + \epsilon_+$ (D' peut être plus grand que 1)
- et il est facile de déterminer un s qui permet d'atteindre ces bornes

Un exemple en base 16

- Quels chiffres ? On choisit $q_i \in \{-10...10\}$
  - on peut décomposer $q_i = q_i^h + q_i^l$ avec
    - $q_i^h \in \{-8, -4, 0, 4, 8\}$
    - $q_i^l \in \{-2, -1, 0, 1, 2\}$
  - C'est le plus grand ensemble pour lequel on peut faire cela
  - En plus il y a un peu de redondance dans cette décomposition
- $q_i D = q_i^h D + q_i^l D$, et chaque terme est juste un décalage de D
- donc archi en deux additionneurs carry-save (dessin)
- et donc on a un additionneur 4 en 2 pour le *prescaling*
- On remarque qu'on a besoin de l'un des produits un peu avant l'autre
- donc on va utiliser la redondance dans la décomposition $q_i = q_i^h + q_i^l$ pour favoriser la sélection d'un des deux chiffres par rapport à l'autre
- *prescaling* : $s \geq 2m - 1$ nous donne $s \geq 7$, prenons $s = 7$
- Reste plus qu'à construire une table à 6 bits d'entrée (le premier bit de D est toujours 1) qui nous donne S en base 4

Un exemple en base 16, suite

- On voit dans cette table que $\frac{8107}{8192} \leq D' \leq \frac{8288}{8192}$
- d'où une fonction de sélection qui ne dépend plus de D.

Pour en finir avec le *prescaling*

- Dans notre graphe P-D, on voit qu'il existe toujours une valeur D_{min} telle que pour $D > D_{min}$ la fonction de sélection ne dépend plus de D
- Il existe aussi toujours une valeur D_{min} telle que pour $D > D_{min}$ on peut approximer D par 1 pour la prédiction de chiffre.
- Il parait qu'il y a des variantes qui font du scaling dans chaque itération

Il y a eu des architectures publiées en base 512 avec un max de prescaling

Considérations architecturales

- Diviseur combinatoire, séquentiel, ou intermédiaire (séquentiel par bloc)
- À cause des dépendances, la version combinatoire n'est pas vraiment plus rapide.

Conversion au vol

- Le quotient arrive poids fort en tête dans un format redondant
- Peut-on éviter la conversion finale avec propagation de retenue ?

Réponse : oui (Ercegovac et Lang), mais je ne détaille pas :

- Maintenir deux candidats quotients
- A chaque itération, les décaler, ajouter un chiffre à chaque, et éventuellement les échanger.

Arrondi au vol

- En virgule flottante, on veut l'arrondi correct de X/D
- Toute l'information nécessaire est dans le dernier reste partiel
- Mais à nouveau, il faut décider son signe, ce qui peut nécessiter une propagation de retenue complète.

Ercegovac et Lang donnent une technique d'arrondi au vol.

Division multiplicative (Newton)

Algo 1 : Newton-Raphson

- Itération de Newton en général : trouve les zéros de $f(x)$ par l'itération :

$$x_{n+1} \leftarrow x_n - \frac{f(x_n)}{f'(x_n)}$$

- On veut calculer l'inverse de D. Annulons $f(x) = \frac{1}{x} - D$
- Il vient

$$R_{n+1} \leftarrow R_n \times (2 - D \times R_n)$$

- Deux multiplications dépendantes dans chaque itération

Convergence quadratique

$$R_{n+1} \leftarrow R_n \times (2 - D \times R_n)$$

- Erreur relative

$$\epsilon_n = \frac{R_n - \frac{1}{D}}{\frac{1}{D}} = DR_n - 1$$

- Il vient

$$\epsilon_{n+1} = -\epsilon_n^2$$

  - Convergence quadratique
  - $\epsilon_n < 0$: approximation par en dessous
  - Convergence si $\epsilon_0 < 1$: approximation initiale de $1/D$
  - On a tout intérêt à obtenir tout de suite R_0 avec 4,8 ou 16 bits
    - Table adressée par quelques bits de D
    - Méthodes d'approximations rapides de fonctions en petite précision : dans un cours ultérieur

Algo 2 : variante utilisée dans l'Itanium

- Soit $R_0 \approx 1/D$
- Soit $\epsilon_0 = 1 - R_0 D$ (c'est l'opposé du précédent)
- Alors
$$\frac{1}{D} = \frac{R_0}{1-\epsilon_0} = R_0(1 + \epsilon + \epsilon^2 + \epsilon^3 + ...)$$
- Première option : Calculer ce polynôme avec des ruses de sioux pour le paralléliser
  - Xemple

$$1 + \epsilon + \epsilon^2 + ... + \epsilon^6 \;=\; 1 + (1 + \epsilon + \epsilon^2)(\epsilon + (\epsilon^2)^2)$$

Algo 3 : variante utilisée dans l'Itanium

$$\frac{1}{D} = \frac{R_0}{1-\epsilon_0} = R_0(1+\epsilon+\epsilon^2+\epsilon^3+...)$$

- Seconde option : approximer la série par ses deux premiers termes
  - Récurrence :

$$\begin{cases} R_0 & \approx \frac{1}{D} \\ \epsilon_n & \leftarrow 1 - R_n \times D \\ R_{n+1} & \leftarrow R_n + R_n \times \epsilon_n \end{cases}$$

  - On a bien $\epsilon_{n+1} = \epsilon_n^2$ comme précédemment
  - Deux FMA dépendants dans l'itération
  - Si le FMA prend 4 cycles pipelinés, possibilité de pipeliner 4 divisions

Cerise sur le gâteau des trois premiers algos

Quid des erreurs d'arrondi ?

- (notre ϵ_n n'est que l'erreur de méthode)

La convergence quadratique les met au carré aussi.

Algo 4 : "Normalisation multiplicative"

- Technique générale :
  - calculer deux récurrences en parallèle
  - faire tendre l'une vers 1 ou 0
  - pour que l'autre tende vers ce que l'on veut
- Pour la division :

$$R = \frac{1}{D} = \frac{1.P_0.P_1.P_2....P_j}{D.P_0.P_1.P_2....P_j} = \frac{N_j}{D_j}$$

  - (N_j comme numérateur, D_j comme dénominateur)
  - On va faire tendre D_j vers 1, alors N_j tendra vers $1/D$
  - $N_j \rightarrow 1/D$, soit (comme d'hab) $\epsilon_j = 1 - N_j D = 1 - D_j$
  - On veut la convergence quadratique : $\epsilon_{j+1} = \epsilon_j^2$
  - ... se réécrit $P_j + 1 = 2 - D_j$

Algo 4 : “Normalisation multiplicative”

$$\begin{cases} P_0 & \approx \frac{1}{D} \\ D_0 & \leftarrow DP_0 \\ N_0 & \leftarrow P_0 \\ N_{j+1} & \leftarrow P_j \times N_j \\ D_{j+1} & \leftarrow P_j \times D_j \\ P_j + 1 & \leftarrow 2 - D_j \end{cases}$$

- Avantages :
  - Deux multiplications en parallèle
  - Avec $N_0 \leftarrow P_0 \times X$ on obtient directement le quotient
- Inconvénient : Les erreurs d'arrondi ne sont pas mises au carré
  - Supposons que $N_{j+1} = (1+\rho)P_j \times N_j$ avec ρ erreur d'arrondi
  - alors en itérant $N_{j+1} = (1+\rho)^j \prod P_k \approx (1+j\rho) \prod P_k$
  - On accumule l'erreur d'arrondi faite à chaque itération
  - (mais bon, il y a peu d'itérations)

Considérations implémentatoires

$$\begin{cases} R_0 & \approx \frac{1}{D} \\ \epsilon_n & \leftarrow 1 - R_n \times D \\ R_{n+1} & \leftarrow R_n + R_n \times \epsilon_n \end{cases}$$

- L'opération $2 - R_j$ peut s'approximer par une inversion bit à bit
  - (avec une erreur égale au bit de poids faible)
  - Avantage : pas de propagation de retenue...
- On n'a pas besoin de multiplications précises au début
  - Idée : utiliser un multiplieur rectangulaire, par exemple 16 bits par 64
  - Premières multiplications en un cycle, puis une en 2 cycles, puis une en 4 cycles

Le diviseur de l'AMD K7

Papier d'Oberman dans Arith 14.

Au prochain numéro

Les divisions avec arrondi correct : JM Muller l'a déjà fait ?
(cf cours sur le flottant)

La racine carrée

De quoi on part

- Utilisé dans les unités flottantes
- on part de $F = M.2^E$ normalisé, et on veut un résultat normalisé
- Si E pair, $\sqrt{F} = \sqrt{M}\ .\ 2^{E/2}$
- Si E impair, $\sqrt{F} = \sqrt{M/2}\ .\ 2^{(E+1)/2}$
- Ainsi on calcule la racine carré d'un nombre $X \in [1/4, 1[$ et le résultat est directement une mantisse normalisée dans $[1/2, 1[$

Racine carrée par récurrence

- Donc on part de $X \in [1/4, 1[$ et on veut calculer S en base β tel que $|\sqrt{X} - S| < \beta^n$
- On va encore calculer $S_j = \sum_{i=1}^{j} s_i \beta^{-i}$ et on aura $S = S_n$
- il faut encore $|\sqrt{X} - S_j| < \rho\beta^{-j}$
- on pose $R_j = \beta^j(X - S_j^2)$
- La récurrence ressemble :

1: $R_0 = X - 1$
2: **for** $j \in \{1..n\}$ **do**
3: $\quad s_{j+1} = \mathrm{Sel}(\beta R_j, S_j)$ ("on intuite" s_{j+1})
4: $\quad R_{j+1} = \beta R_j - 2s_{j+1}S_j - s_{j+1}^2\beta^{-j-1}$
5: **end for**

- S_j a pris la place de D.
- Le calcul de R_{j+1} est en deux additions

Intervalles de sélection

- $L_k(j) = 2S_j(k-\rho) + (k-\rho)^2\beta^{-j-1}$
- $U_k(j) = 2S_j(k+\rho) + (k+\rho)^2\beta^{-j-1}$
- On peut faire un graphe P-D (qui a de moins en moins de raisons de s'appeller un graphe P-D).
- Comme pour la division, $\mathrm{Sel}(\beta R_j, S_j)$ va utiliser des valeurs tronquées de βR_j et S_j
- Damned, la sélection de s_{j+1} dépend directement de j
- Heureusement, le terme décroît vite et on arrivera à avoir une fonction unique pour tous les $j \geq j_0$ (avec j_0 petit)
  (calculer pour chaque k $\min_j U_k(j)$ et $\max_j L_k(j)$)
- On montre que $j_0 = 0$ est possible uniquement en base 2
- Ensuite
  - soit on a une petite table qui donne les j_0 bits de S_{j_0} à partir de quelques bits de poids forts de X,
  - soit on bricole pour avoir une fonction de sélection uniforme sur toutes les itérations.

Des bricolages à la Ercegovac

- On se place en base 4, {-2...2}, R_j en carry-save : on arrive à trouver une fonction de sélection unique pour $j \geq j_0 = 3$
- Ensuite on regarde $j = 0, j = 1, j = 2$: il y a peu de valeurs possibles de S_0, S_1 et S_2
  - On doit pouvoir représenter toutes les valeurs de $s \in [1/2, 1[$, cela impose $s_0 = 1$
  - Du coup si $s_1 > 0$ on aura $s > 1$: donc $s_1 \in \{-2..0\}$.
  - Du coup il n'y a que trois valeurs possibles de de S_1
  - ...
  - Arrivé à $j = 2$ la combinatoire commence à peser
  - et on arrive à bricoler une fonction de sélection qui couvre tous les j quand même, en remplaçant la troncature de S_j par une fonction ad-hoc simple qui marche aussi bien
  - Ce bricolage a été publié par Ercegovac et Lang en 90.

Racine carrée multiplicative

- On veut calculer $\sqrt{S}$. Annulons $f(x) = x^2 - S$
- Il vient
$$S_{n+1} \leftarrow 0.5(S_n + \frac{x}{Sn})$$
- Zut, une division
- Calculons plutôt la racine carrée inverse : $\frac{1}{\sqrt{S}}$
- (on multipliera ensuite par S)
- Annulons $f(x) = 1/x^2 - R$
- ... d'où
$$S_{j+1} = \frac{1}{2} S_j (3 - S S_j^2)$$
- 3 multiplications dépendentes.
- Algo par normalisation : 3 multiplications dont 2 dépendentes (voir le Ercegovac et Flynn)

Le stage de DEA de Jérémie Detrey

Objectifs

- Des outils pour comparer sur pièces les arithmétiques LNS et flottantes pour les applis de traitement du signal sur FPGA
  - Précision et dynamique des nombres
  - Opérateurs nécessaires à l'application à implémenter
- En pratique : des bibliothèques d'opérateurs entièrement paramétrables
- La comparaison n'est équitable que si on a les meilleurs opérateurs existant dans chaque camp
- Effet de bord : des bibliothèques performantes

Ce que Jérémie a fait, aucune bête ne l'aurait fait

- Jérémie a écrit plein de diviseurs et de racines carrées en base 2, 4 et 8, avec différents ensembles de chiffres
  - FPGA donc les restes partiels sont non redondants *(fast carry)*
  - Division : base 4, chiffres {-3,...3}
  - Racine carrée : base 2, chiffres {-1,1} (racine carré restaurante ?)
- Il a aussi fait de la vraie recherche sur les opérateurs LNS, mais c'est pour une autre fois.

Pas de conclusion

Oh, un client

Avec une bonne méthode d'approximation initiale de $1/D$ ou $\sqrt{S}$ on pourra gagner des itérations de Newton-Raphson.