Opérateurs arithmétiques matériels
Fonctions continues

Florent de Dinechin

7 décembre 2005

Introduction

À quoi sert-ce ?

Utilisation de plus en plus fréquente de fonctions de calcul “compliquées” dans les architectures et en particulier dans les FPGA.

Exemples :

- traitement du signal (filtres non linéaires, ondelettes, . . .)
- traitement d’images (normes, rotations, courbes, application de textures, projections 3D $\longrightarrow$ 2D, . . .)
- contrôle (changements de repères, fonctions non linéaires, . . .)
- constructions d’autres opérateurs arithmétiques
  - addition LNS,
  - addition complexe en représentation polaire (?)

Évaluation d'une fonction : schéma général

On divise l'évaluation en trois phases :

réduction d'argument	$(f, x) \xrightarrow{RA} (g, x^*)$
évaluation	$y^* = g(x^*)$
arrondi et post-traitement	$y^* \xrightarrow{APT} y$

Réduction d'argument additive

$$x^* = x - kC$$

Exemple : sinus

1. $x \xrightarrow{RA} (x^*, k)$ tels que $\begin{cases} x^* \in [0, 2\pi] \\ x = x^* + 2k\pi \end{cases}$
2. $\sin(x^*) \xrightarrow{APT} \sin(x) = \sin(x^*)$

La précision des calculs intermédiaires est importante !

La spécification doit être précise (attention au complément à 2 non symétrique)

Réduction d'argument multiplicative

$$x^* = x/C^k$$

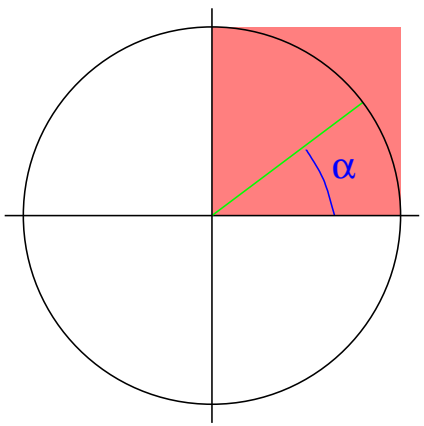
Exemple : le logarithme

1. $x \xrightarrow{RA} (x^*, k)$ tels que $\begin{cases} x^* \in [1/2, 2] \\ x^* = x/2^k \end{cases}$
2. $\ln(x^*) \xrightarrow{APT} \ln(x) = \ln(x^*) + k \ln(2)$

La précision des calculs intermédiaires est importante !

Autres réductions d'argument

- par intervalles
- en utilisant les propriétés de la fonction : exemple, on peut évaluer sinus et cosinus sur $[0, 2\pi]$ en n'ayant que l'évaluation de sinus sur $[0, \pi/2]$:



$$\alpha \in [0, \pi/2[\longrightarrow \begin{cases} \alpha' = \alpha \\ \sin(\alpha) = \sin(\alpha') \\ \cos(\alpha) = \cos(\alpha') = \sin(\frac{\pi}{2} - \alpha') \end{cases}$$

$$\alpha \in [\pi/2, \pi[\longrightarrow \begin{cases} \alpha' = \alpha - \pi/2 \\ \sin(\alpha) = \sin(\frac{\pi}{2} - \alpha') \\ \cos(\alpha) = -\sin(\alpha') \end{cases}$$

$$\alpha \in [\pi, 3\pi/2[\longrightarrow \begin{cases} \alpha' = \alpha - \pi \\ \sin(\alpha) = -\sin(\alpha') \\ \cos(\alpha) = -\cos(\alpha') = -\sin(\frac{\pi}{2} - \alpha') \end{cases}$$

$$\alpha \in [3\pi/2, 2\pi[\longrightarrow \begin{cases} \alpha' = \alpha - 3\pi/2 \\ \sin(\alpha) = -\cos(\alpha') = -\sin(\frac{\pi}{2} - \alpha') \\ \cos(\alpha) = \sin(\alpha') \end{cases}$$

Discrétisation du domaine et de l'image

- Fonction mathématique $f : [ab] \rightarrow [cd]$
- Fonction discrète $F : \{0...2^n - 1\} \rightarrow \{0...2^m - 1\}$
- La correspondance exacte dépend du contexte (réduction d'argument, besoins de l'application)
- Exemple : le sin/cos précédent
  - en entrée,

$$\begin{aligned} 0 &\leftrightarrow 0 \\ 2^n - 1 &\leftrightarrow \pi/2(1 - 2^{-n}) \end{aligned}$$

    vérifier que la réduction d'argument réduit $\pi/2$ en 0, etc.
  - en sortie,

$$\begin{aligned} 0 &\leftrightarrow 0 \\ 2^m - 1 &\leftrightarrow 1 \end{aligned}$$

    On ne va jamais utiliser la plus petite valeur négative (le complément à 2 n'est pas symétrique)
- Exercice : contruire le matériel pour la réduction du sinus
- Exemple de discrétisation différente pour $f : [ab] \rightarrow [cd]$:

$$\begin{aligned} 0 &\leftrightarrow a + 2^{-n-1}(b - a) \\ 2^n - 1 &\leftrightarrow b - 2^{-n-1}(b - a) \end{aligned}$$

Sources d'erreur d'une approximation

Il y a deux sources d'erreur dans une évaluation :

1. précision de l'approximation d'une fonction par une autre (erreur de méthode)
2. calculs en précision finie (erreurs d'arrondis)

Les deux sources d'erreur cumulent en général leurs effets...

... et les gens qui en escamotent une sont des escrocs.

On peut ajouter une erreur de quantification :
que représente un nombre discret ?

- lui-même uniquement ?
- un intervalle de réels qui l'entoure ?

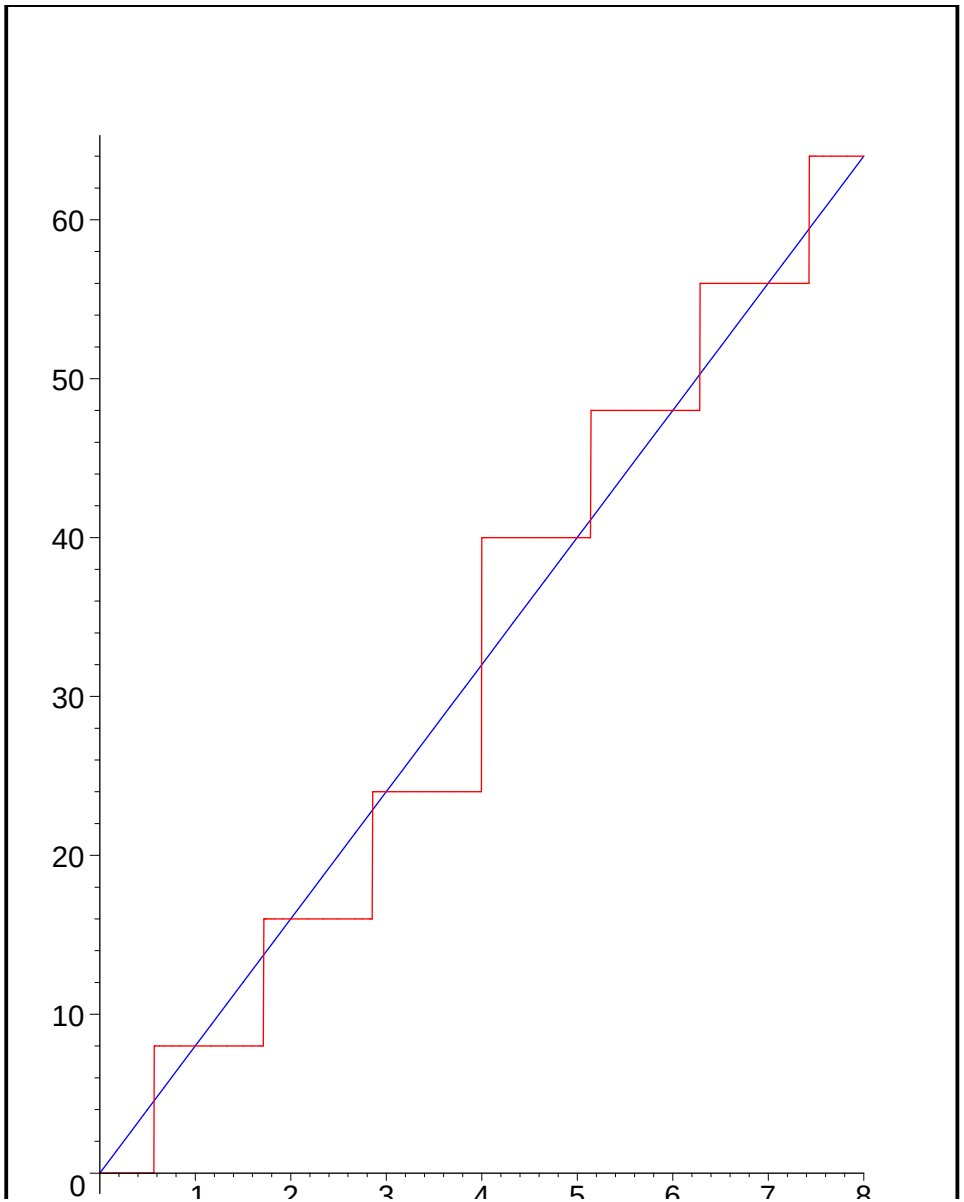
Cela aussi fait partie de la spécification

Précision de l'approximation (erreur de méthode)

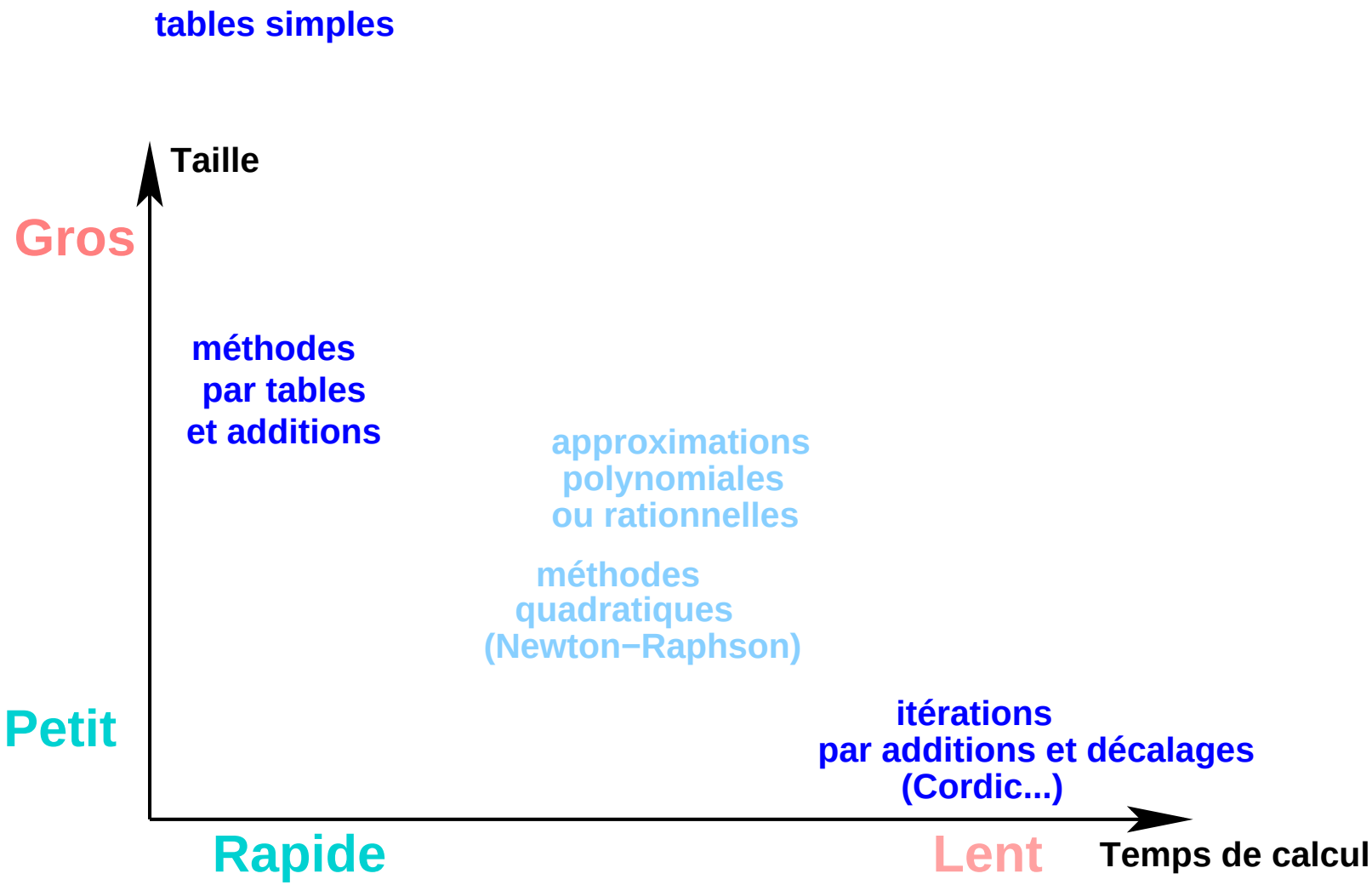
On va en voir plein

Calculs en précision finie (erreurs d'arrondis)

Exemple : discrétisation d'une bête droite.



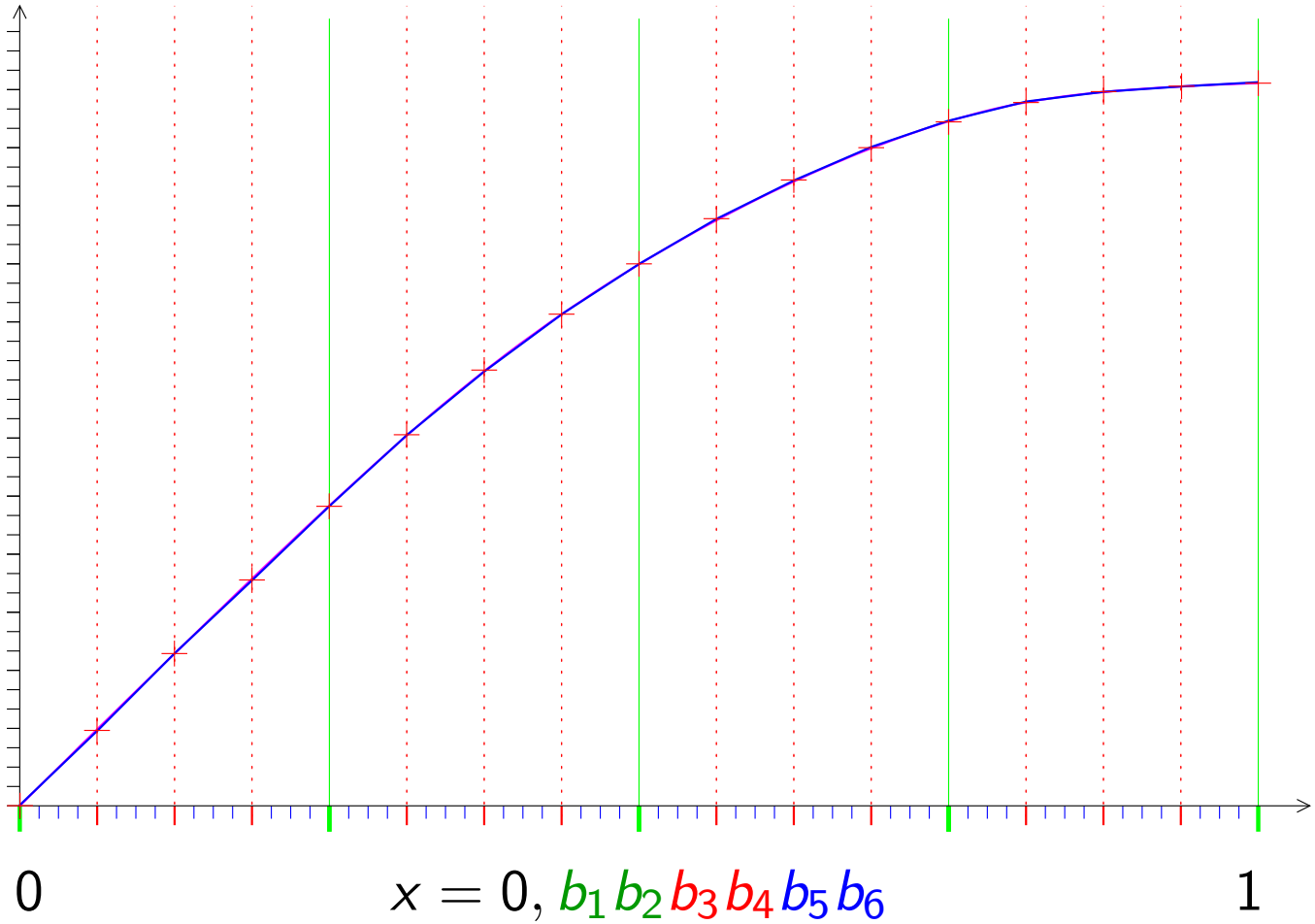
Évaluation des fonctions : méthodes



Méthodes multipartites

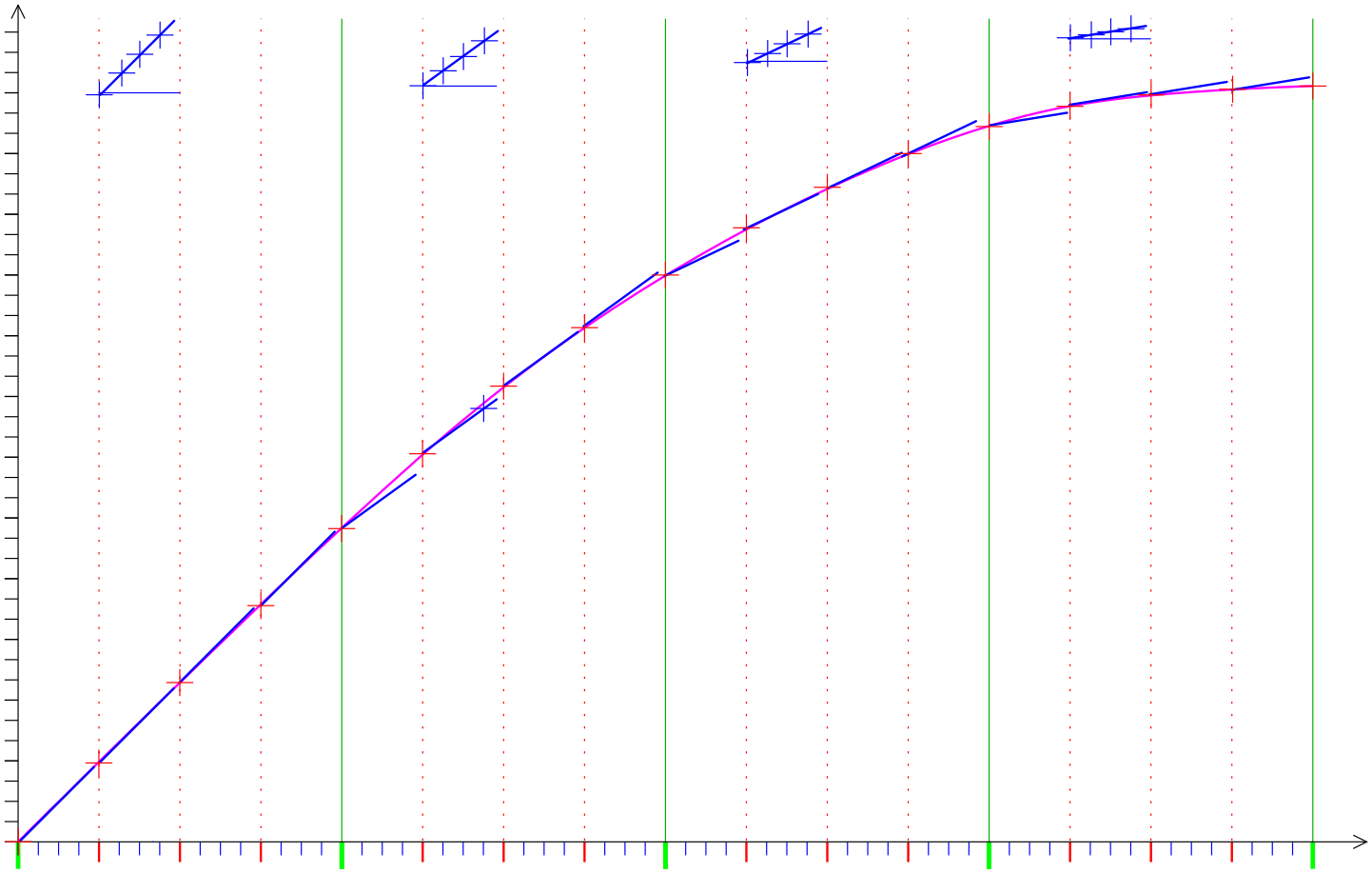
Principe de la méthode bipartite

approximation linéaire par morceaux



Principe de la méthode bipartite

pour laquelle on fixe la pente constante sur les intervalles verts.



$x = 0, b_1\, b_2\, b_3\, b_4\, b_5\, b_6$

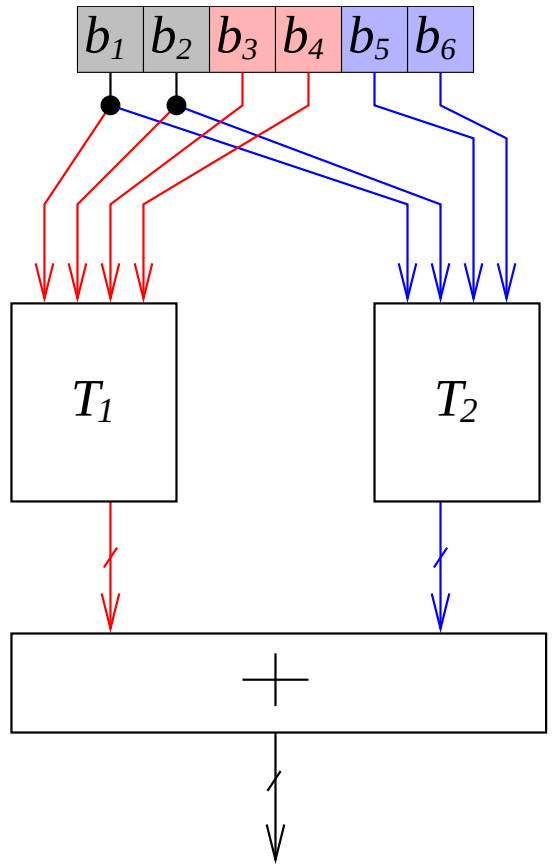
L'architecture bipartite

Deux tables :

- les $+$ (16 valeurs)
- les $+$ (4×4=16 valeurs)

et un additionneur.
Voilà qui remplace une table de 64 valeurs.

Dans le cas général, on comprime
une table de 2^n valeurs
en deux tables de $2^{2n/3}$ valeurs chacune.



Pour les amateurs de maths...

Tout cela c'est Taylor et compagnie.
Posons $k = n_1 = n_2 = n_3 = n/3$.
Écrivons $x = 0, b_1 b_2 ... b_n$ sous la forme $x_1 x_2 x_3$;

$$x = x_1 + x_2 \times 2^{-k} + x_3 \times 2^{-2k} \quad \text{avec} \quad \begin{cases} x_1 = 0, b_1 ... b_k \\ x_2 = 0, b_{k+1} ... b_{2k} \\ x_3 = 0, b_{2k+1} ... b_n \end{cases}$$

On utilise un développement de Taylor d'ordre 1 au point $x_1 + x_2 \times 2^{-k}$:

$$f(x) = f(x_1 + x_2 2^{-k}) + x_3 2^{-2k} f'(x_1 + x_2 2^{-k}) + \epsilon_1$$

Puis on approche $f'(x_1 + x_2 2^{-k})$ par un développement d'ordre 0 en x_1 :

$$f'(x_1 + x_2 2^{-k}) = f'(x_1) + \epsilon_2$$

Tout cela c'est Taylor et compagnie.

Et finalement :

$$f(x) = f(x_1 + x_2 2^{-k}) + x_3 2^{-2k} f'(x_1) + \epsilon_1 + \epsilon_2$$

Pour implémenter, on va essayer de

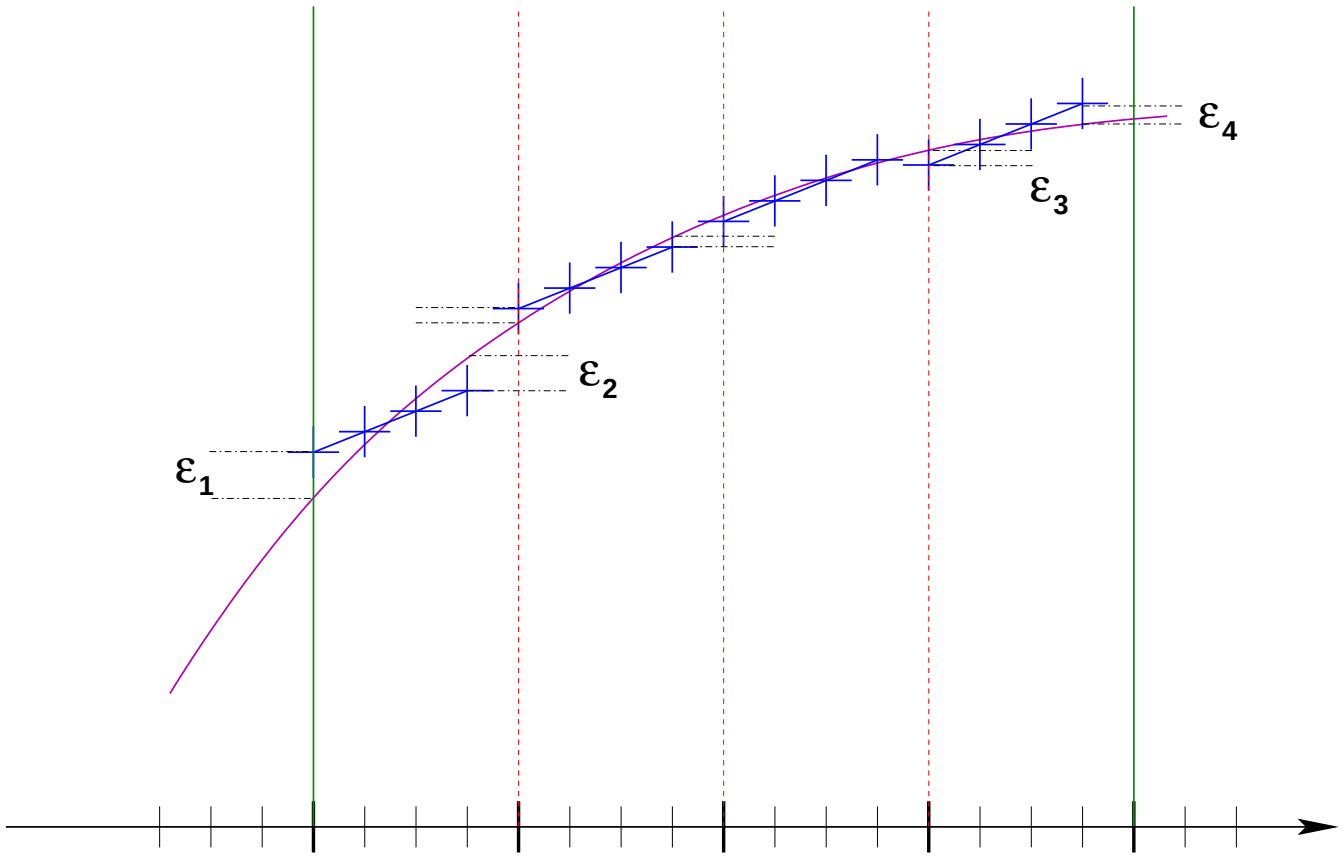
- minimiser ϵ_1 et ϵ_2,
- et tenir compte des erreurs d'arrondi
  (car ϵ_1 et ϵ_2 ne représentent que l'erreur de méthode).

Remarques

- Taylor est pessimiste ! on arrivera à un sinus en (3,4,5) au lieu de (4,4,4) pour 12 bits
- Taylor est rigide ! Keskon fait si n n'est pas divisible par 3 ? Si on veut un bit de plus en sortie qu'en entrée ?
- donc désormais on revient à des petits dessins, par exemple TSVP

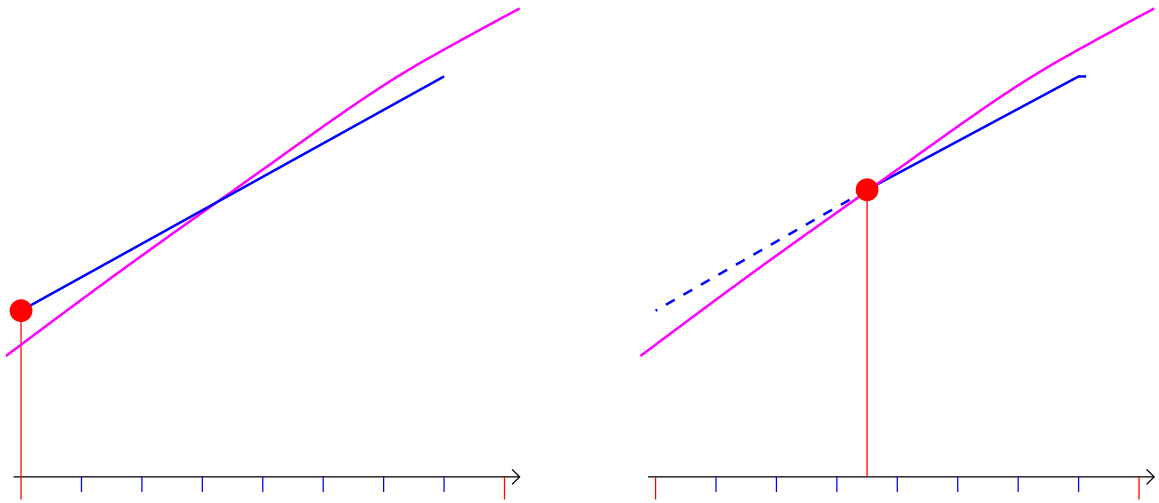
Calcul de l'erreur dans les tables bipartites

Sous une hypothèse de convexité, on peut calculer exactement les valeurs des tables qui minimisent l'erreur de méthode :



Il suffira d'écrire $\varepsilon_1 = -\varepsilon_2 = -\varepsilon_3 = \varepsilon_4$, et on obtiendra la pente et la valeur de la première table

D'où l'idée de la symétrie (Schulte et Stine)

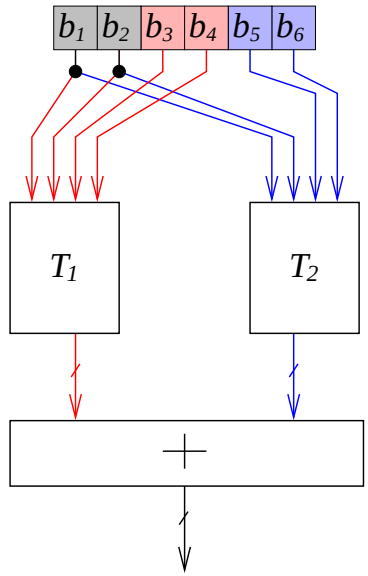


Gain sur la seconde table :

- un bit sur l'adresse ;
- un bit sur la sortie.

Par contre il faut à présent gérer les signes : on verra quand on en sera à parler de nombres discrets.

Quant aux erreurs d'arrondi...



- Si on remplit les deux tables avec des valeurs arrondies au dernier bit,
  - Au pire cas l'erreur d'arrondi est alors d'un bit complet (deux demi-bits)
  - plus l'erreur de méthode, donc on n'aura pas l'arrondi fidèle.

Bits de garde

- On doit donc remplir les tables avec une précision supérieure de g bits
  (bits de garde)...
  - l'erreur d'arrondi des tables est alors aussi petite que l'on veut (2.2^{-g} bits)
  - mais alors il faut arrondir la somme à la précision de sortie (encore une erreur d'un demi-bit au pire).
  - donc on n'aura pas l'arrondi au plus près : c'est la malédiction correspondant au dilemme du fabricant de tables (TMD)

Ruse de sioux de Matula

- remplir nos tables pour mettre un 1 implicite après le dernier bit de garde de la somme
- alors l'erreur de l'arrondi final est $1/2 - 2^{-g-1}$ au lieu de $1/2$.
- Exemple : $g = 2$:
  - 0,00 s'arrondit en 0,00 (erreur 0)
  - 0,01 s'arrondit en 0,00 (erreur 0,01)
  - 0,10 s'arrondit en 1,00 (erreur 0,10, max)
  - 0,11 s'arrondit en 1,00 (erreur 0,01)
- Avec un bit et avec un $g+1$-ième bit implicite
  - 0,00 représente 0,001, s'arrondit en 0,000 (erreur 0,001)
  - 0,01 représente 0,011, s'arrondit en 0,000 (erreur 0,011)
  - 0,10 représente 0,101, s'arrondit en 1,000 (erreur 0,011)
  - 0,11 représente 0,111, s'arrondit en 1,000 (erreur 0,001)

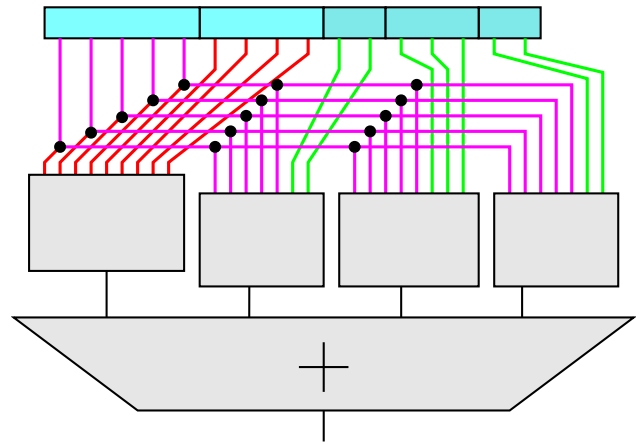
Le truc c'est juste que l'erreur max est désormais symétrique.

Two multipartite methods

Split x_3 *again and distribute* $x_3 f'(x_1)$ *:*

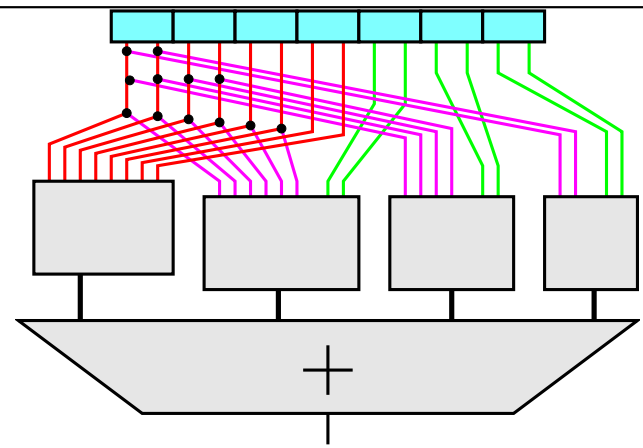
J.E. Stine and M.J. Schulte. The symmetric table addition method for accurate function approximation. *Journal of VLSI Signal Processing*, 21(2) :167–177, 1999.

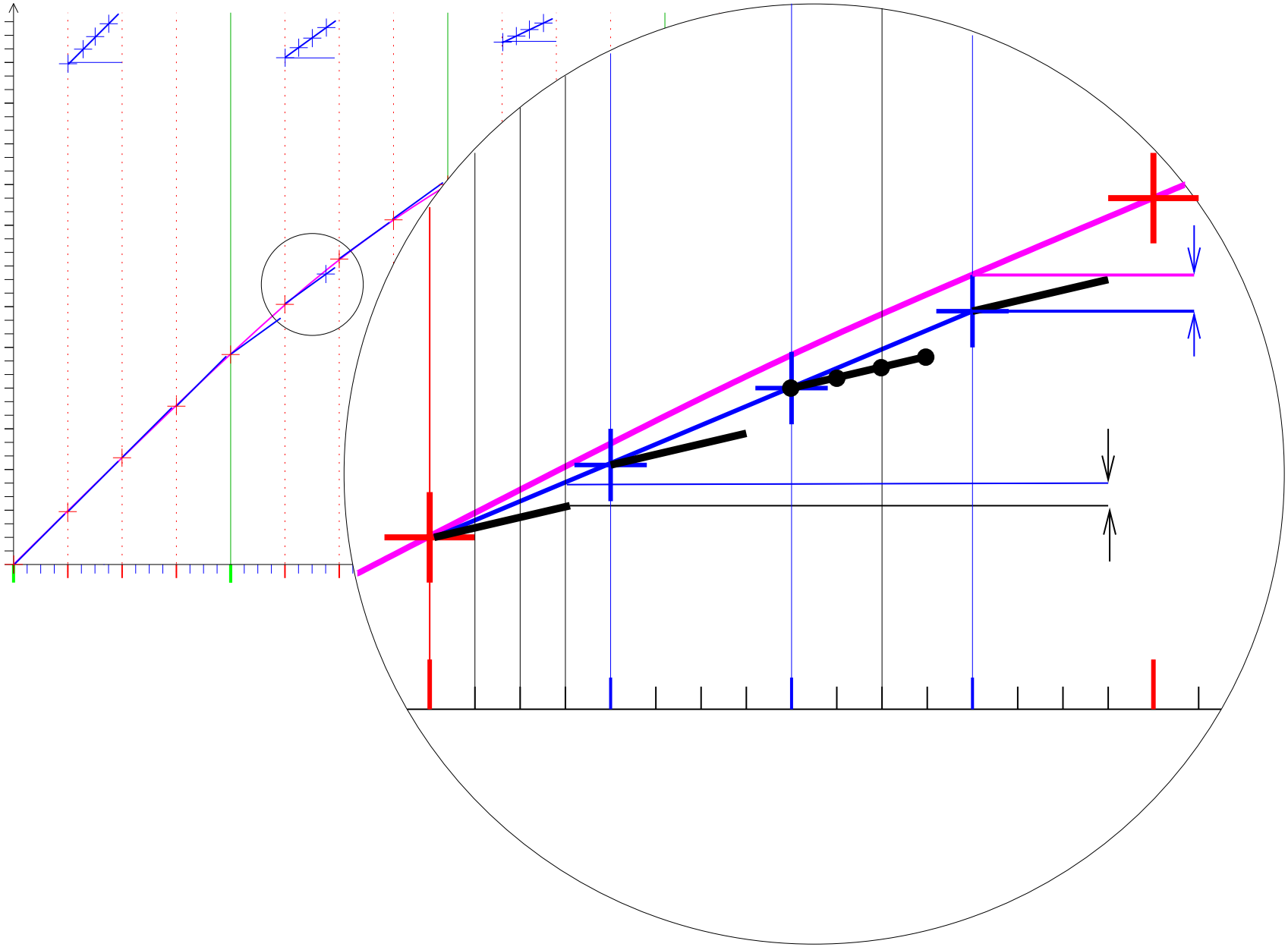
(Symmetry not shown)



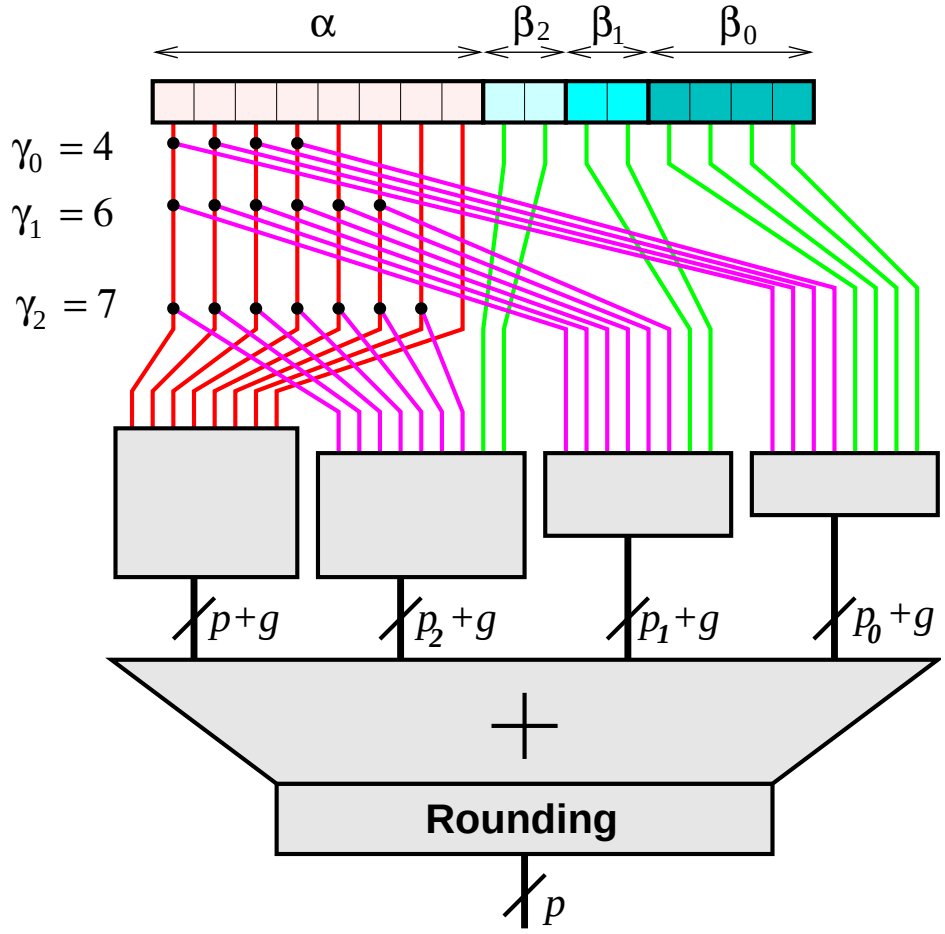
Take a rougher slope for the finer bits to balance approximation errors :

J.M. Muller. A few results on table-based methods. *Reliable Computing*, 5(3) :279–288, 1999.

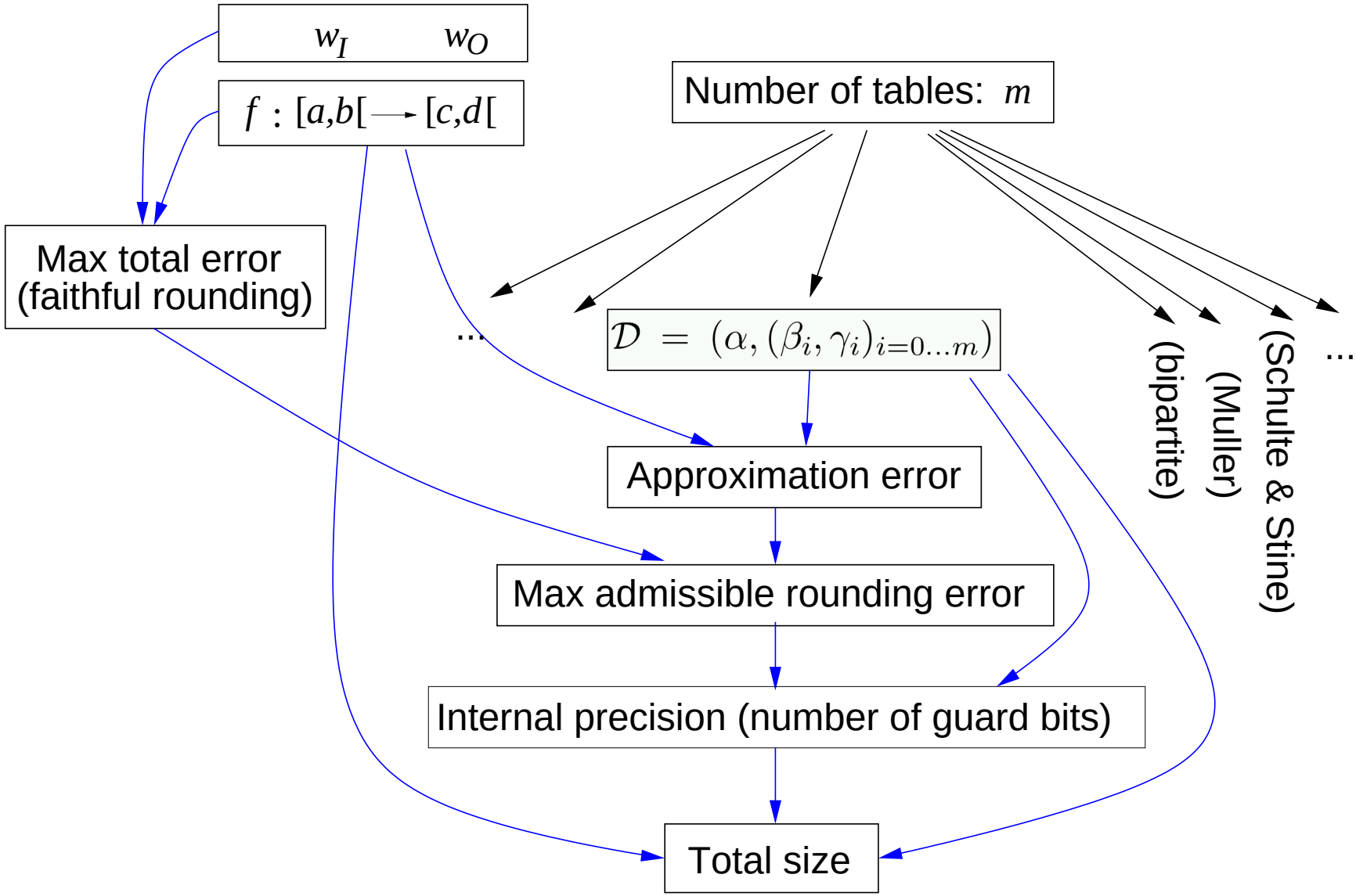




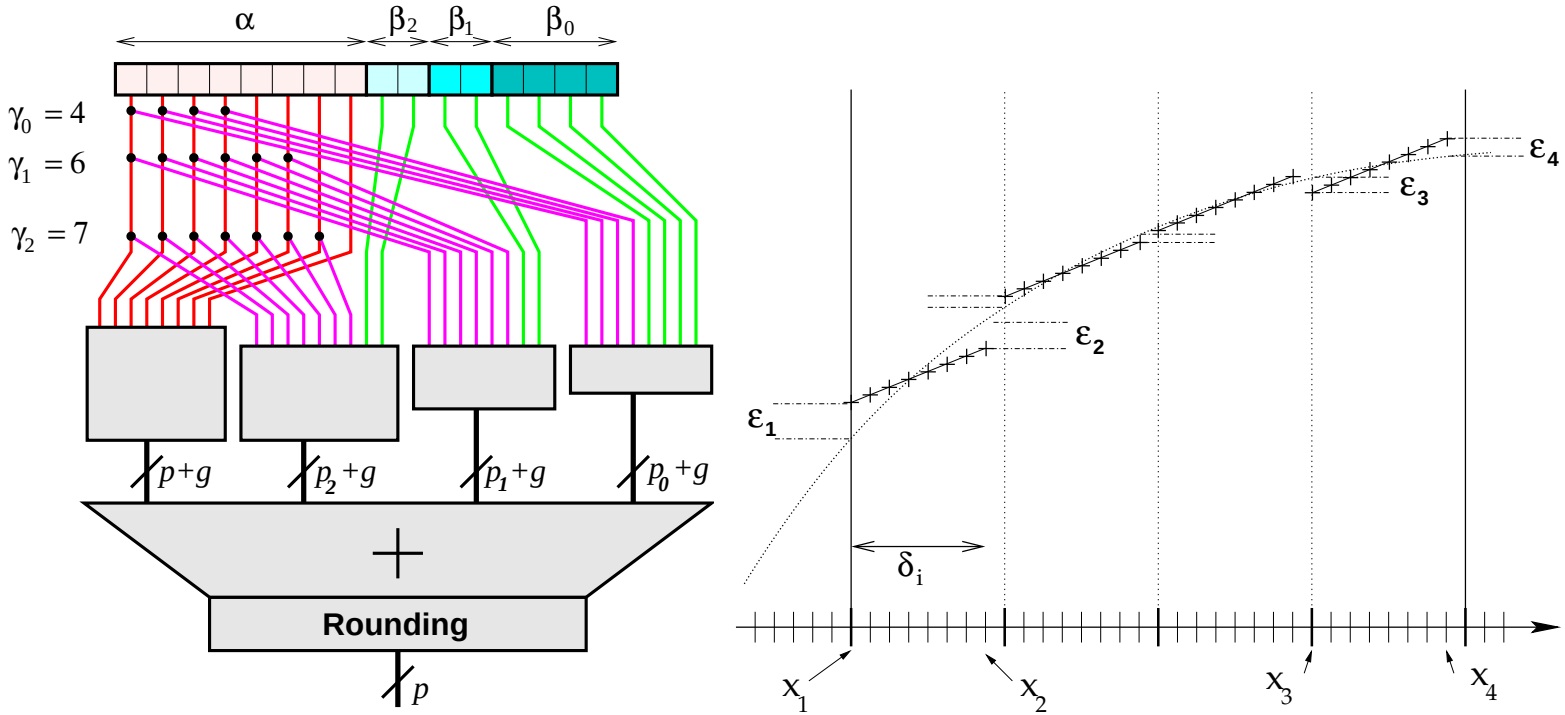
The truth is in between



Generalized multipartite method



Computing the approximation error



B_i and C_i are the input words to the TOi

$$\mu_i(B_i) = a + (b-a)2^{-w_I+p_i}B_i. \tag{1}$$

$$\epsilon^{\mathcal{D}}_{\text{approx}} = \sum_{i=0}^{m-1} \epsilon^{\mathcal{D}}_i \tag{2}$$

$$\epsilon^{\mathcal{D}}_i = \max(|\epsilon^{\mathcal{D}}_i(0)|, |\epsilon^{\mathcal{D}}_i(2^{\gamma_i}-1)|) \tag{3}$$

$$\varepsilon_1 = -\varepsilon_2 = -\varepsilon_3 = \varepsilon_4 \;=\; \epsilon^{\mathcal{D}}_i(C_i) \tag{4}$$

$$s^{\mathcal{D}}_i(C_i) = \frac{f(x_2) - f(x_1) + f(x_4) - f(x_3)}{2\delta_i} \tag{5}$$

$$\epsilon^{\mathcal{D}}_i(C_i) = \frac{f(x_2) - f(x_1) - f(x_4) + f(x_3)}{4} \tag{6}$$

$$\delta_i = (b-a)2^{-w_I+p_i}(2^{\beta_i}-1) \quad (7)$$

$$x_1 = a + (b-a)2^{-\gamma_i} C_i \quad (8)$$

$$x_2 = x_1 + \delta_i \quad (9)$$

$$x_3 = x_1 + (b-a)(2^{-\gamma_i} - 2^{-w_I+p_i+\beta_i}) \quad (10)$$

$$x_4 = x_3 + \delta_i \quad (11)$$

$$\epsilon_i^{\mathcal{D}} = \max(|\epsilon_i^{\mathcal{D}}(0)|, |\epsilon_i^{\mathcal{D}}(2^{\gamma_i}-1)|) \quad (12)$$

Computing the sizes

We need the number of guard bits :

$$g = \left\lceil -w_O - 1 + \log_2 \frac{(d-c)m}{(d-c)2^{-w_O-1} - \epsilon_{\text{approx}}} \right\rceil \tag{13}$$

... and the range of each table :

$$r_i = \max(|s_i(0) \times \delta_i|, |s_i(2^{\gamma_i} - 1) \times \delta_i|) \tag{14}$$

... to compute its output width :

$$w_i = \lceil w_O + g + \log_2(r_i/(d-c)) \rceil \tag{15}$$

Filling the tables (using symmetry)

First compute the real values :

$$x_l = a + (b-a)2^{-\alpha}A \tag{16}$$

$$x_r = x_l + \sum_{i=0}^{m-1} \delta_i \tag{17}$$

$$\widetilde{\mathrm{TIV}}(A) = \frac{f(x_l) + f(x_r)}{2} \tag{18}$$

$$\widetilde{\mathrm{TO}_i}(C_i B_i) = s(C_i) \times 2^{-w_l + p_i}(b-a)(B_i + \frac{1}{2}) \tag{19}$$

Filling the tables (using symmetry)

Then round :

$$\mathrm{TO}_i(C_iB_i) = \left\lfloor \frac{2^{w_O+g}}{d-c}\widetilde{\mathrm{TO}}_i(C_iB_i) \right\rfloor \tag{20}$$

If m is odd :

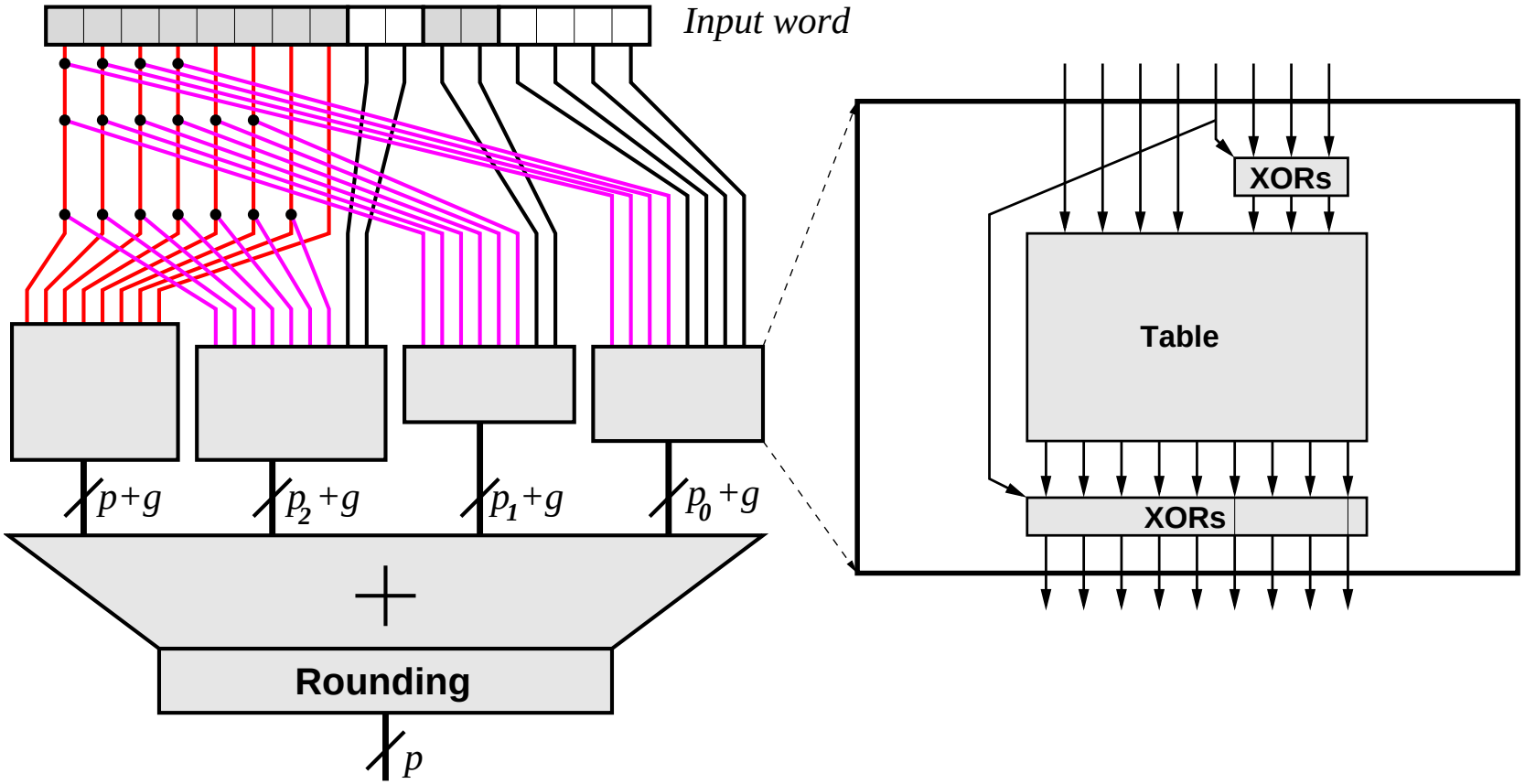
$$\mathrm{TIV}(A) = \left\lfloor 2^{w_O+g} \times \frac{\widetilde{\mathrm{TIV}}(A) - c}{d-c} + \frac{m-1}{2} + 2^{g-1} \right\rfloor \tag{21}$$

and if m is even :

$$\mathrm{TIV}(A) = \left\lfloor 2^{w_O+g} \times \frac{\widetilde{\mathrm{TIV}}(A) - c}{d-c} + \frac{m}{2} + 2^{g-1} \right\rfloor \tag{22}$$

(C'est pour avoir le 1 implicite)

Archi



Résultats

Rappel : $16 \times 2^{16} = 1\ 048\ 576$

Sinus, 16 bits

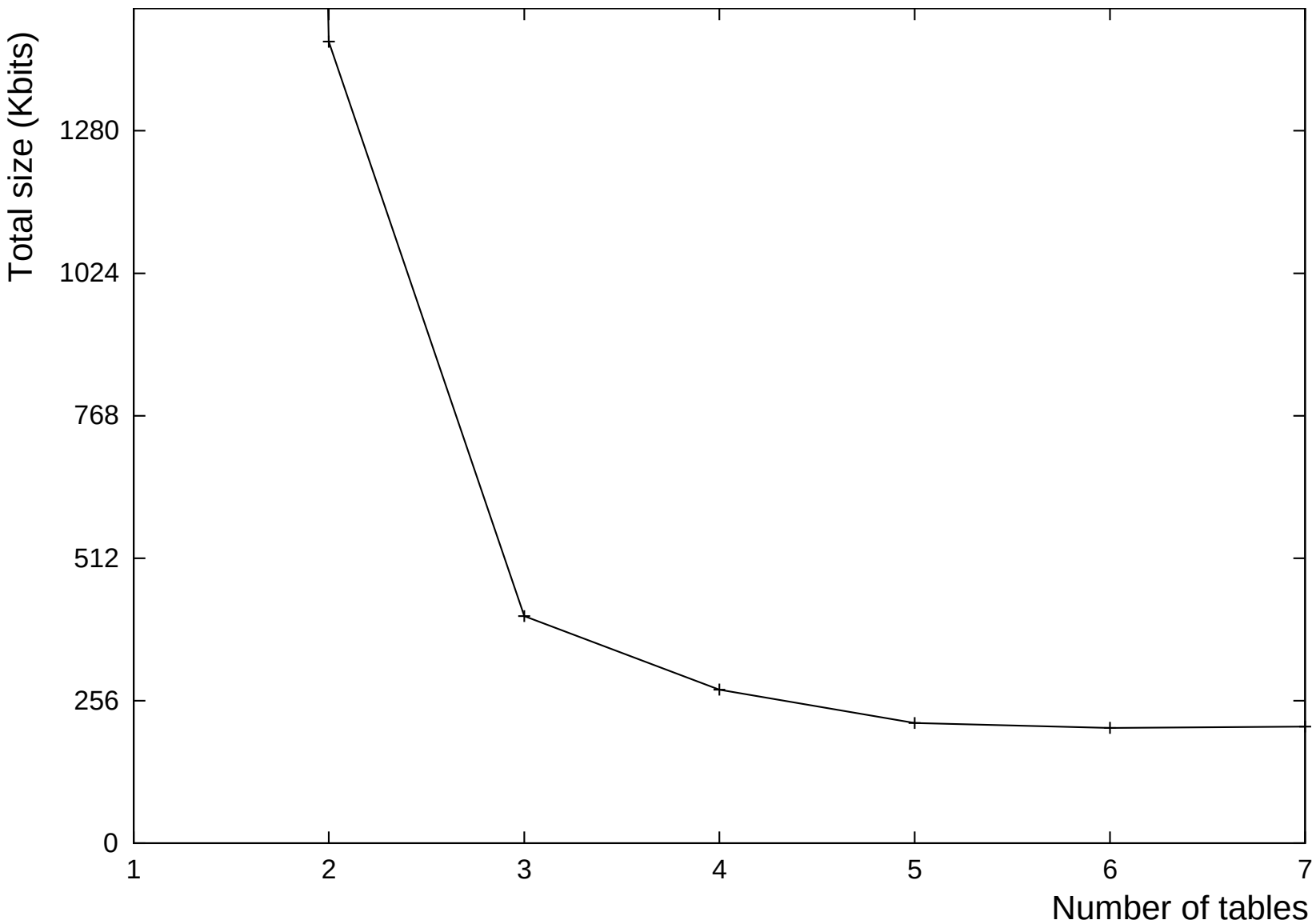
m	α	β	γ_i	β_i	size
1	10	6	5	6	$17.2^{10} + 7.2^{10} = 24576$
2	8	8	7,4	3,5	$19.2^{8} + 10.2^{9} + 8.2^{8} = 12032$
3	8	8	7,6,4	2,3,3	$18.2^{8} + 9.2^{8} + 7.2^{8} + 4.2^{6} = 8960$
4	8	8	7,5,3,3	2,2,2,2	$19.2^{8} + 10.2^{8} + 8.2^{6} + 6.2^{5} + 4.2^{5} = 8256$

2^x, 16 bits

m	α	β	γ_i	β_i	size
1	10	6	5	6	$16.2^{10} + 6.2^{10} = 22528$
2	8	8	7,4	3,5	$17.2^{8} + 9.2^{9} + 6.2^{8} = 10496$
3	8	8	7,6,4	2,2,4	$17.2^{8} + 9.2^{8} + 7.2^{7} + 5.2^{7} = 8192$
4	8	8	7,6,5,4	2,2,2,2	$18.2^{8} + 10.2^{8} + 8.2^{7} + 6.2^{6} + 4.2^{5} = 8704$

En plus les tables plus petites sont plus rapides

Sine for 24 bits



Virtex implementation (16 bits)

f	m	#LUTs	delay	T_{synth}	CF
sin	1	1375	43 ns	122 s	19.4
	2	799	42 ns	50 s	16.4
	3	628	39 ns	34 s	15.6
	4	664	41 ns	37 s	14.3
2^x	1	1839	43 ns	206 s	16.7
	2	959	39 ns	67 s	16.6
	3	864	42 ns	52 s	14.5
	4	801	39 ns	54 s	14.4

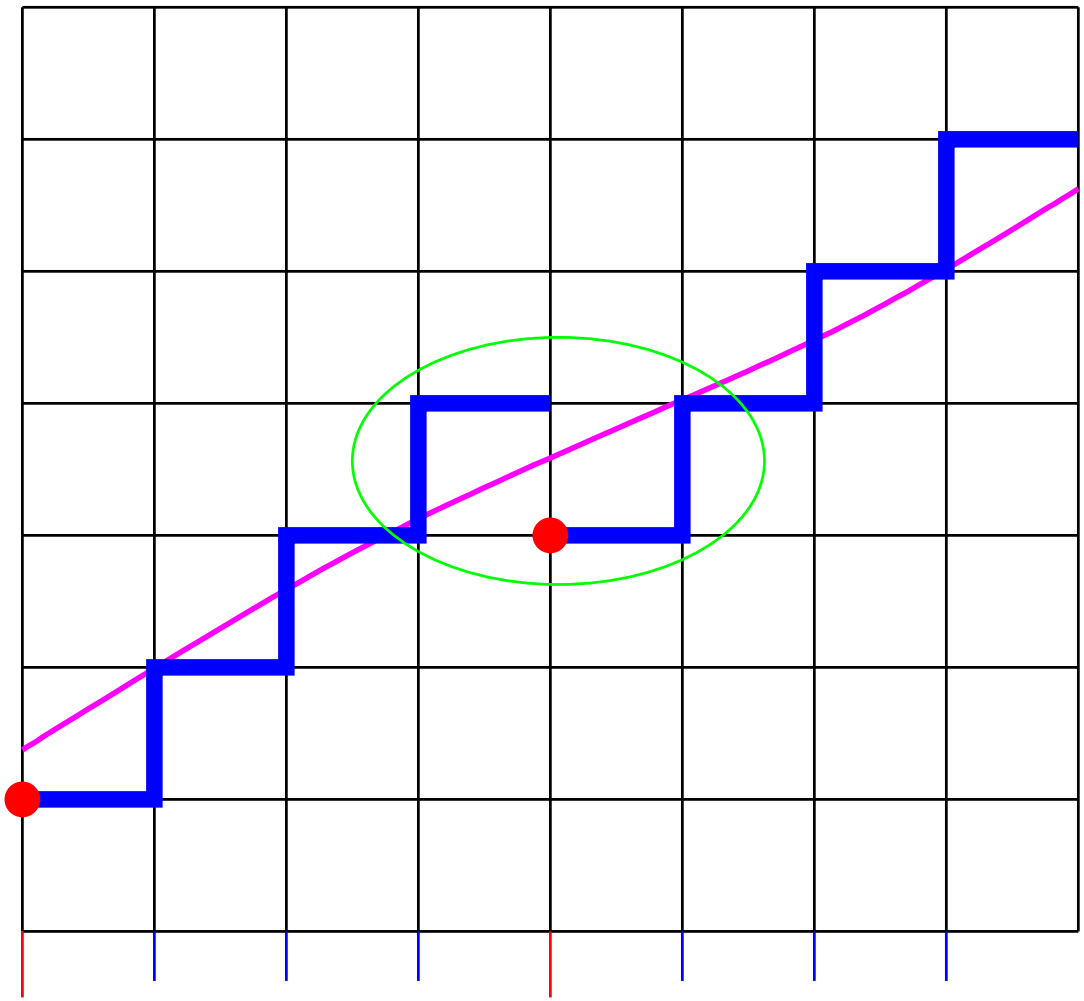
Current Virtex chips hold between 1,536 and 64,896 LUTs
(XCV50 and XCV3200E respectively)

Le retour de Jérémie Detrey

... qui nous raconte cette fois-ci son stage de MIM1.

Limits of the method

- Non-monotonicities

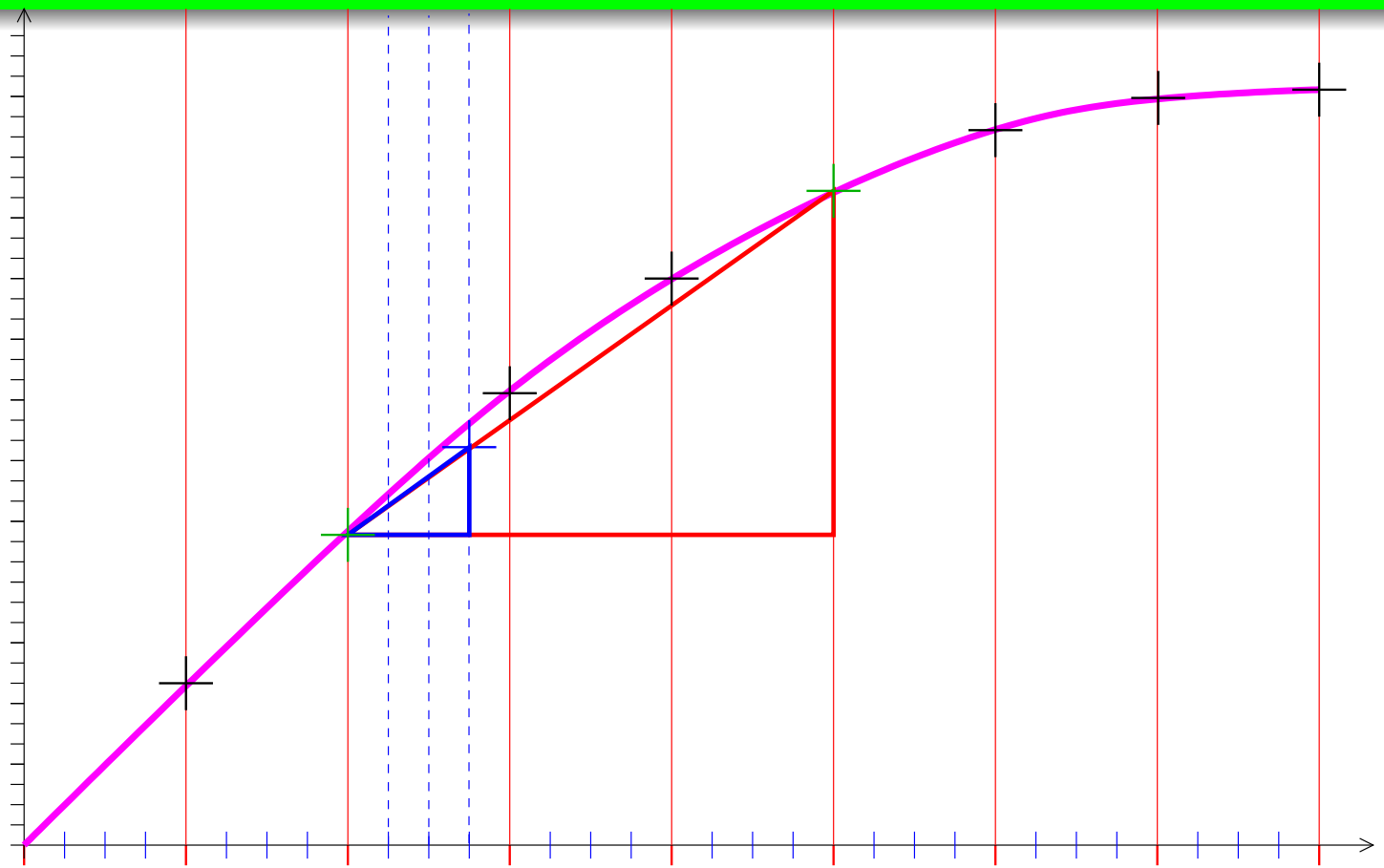


  - (never more than 1 bit)
  - can be solved by considering convexity
- 1-bit overflows a real problem

Les méthodes ATA

Les méthodes précédentes sont des lectures de tables suivies d'additions.
Les méthodes ATA (addition, table, addition) se permettent des additions avant les lectures de tables.

Méthode ATA du premier ordre



$x = a_2\, a_1\, a_0\, b_1\, b_0$

Principe : homothétie de rapport 2^k entre les triangles bleus et rouges.

Algo : si $x = a_{\alpha-1}...a_0 b_{\beta-1}...b_0$, avec forcément $\alpha > \beta$

- une addition sur α bits pour calculer $a_{\alpha-1}...a_0 + b_{\beta-1}...b_0$
- deux lectures de la même table,
  - $f(a_{\alpha-1}...a_0)$ et
  - $f(a_{\alpha-1}...a_0 + b_{\beta-1}...b_0)$,

  en parallèle, ou en pipeline (avant et après l'addition)
- une soustraction pour calculer le Δy rouge (sur moins de α bits),
- un décalage de β bits pour calculer le Δy bleu correspondant,
- une addition finale.

À noter que

- l'erreur est toujours meilleure que dans le cas bipartite,
- on peut bricoler ici aussi pour diviser l'erreur par deux...

Tout cela c'est Taylor et compagnie

- Soit à calculer $f(A + 2^{-\beta}B)$
- On utilise l'approximation

$$f(A + 2^{-\beta}B) \approx f(A) \; + \; 2^{-\beta}Bf'(A)$$

- Et on approxime

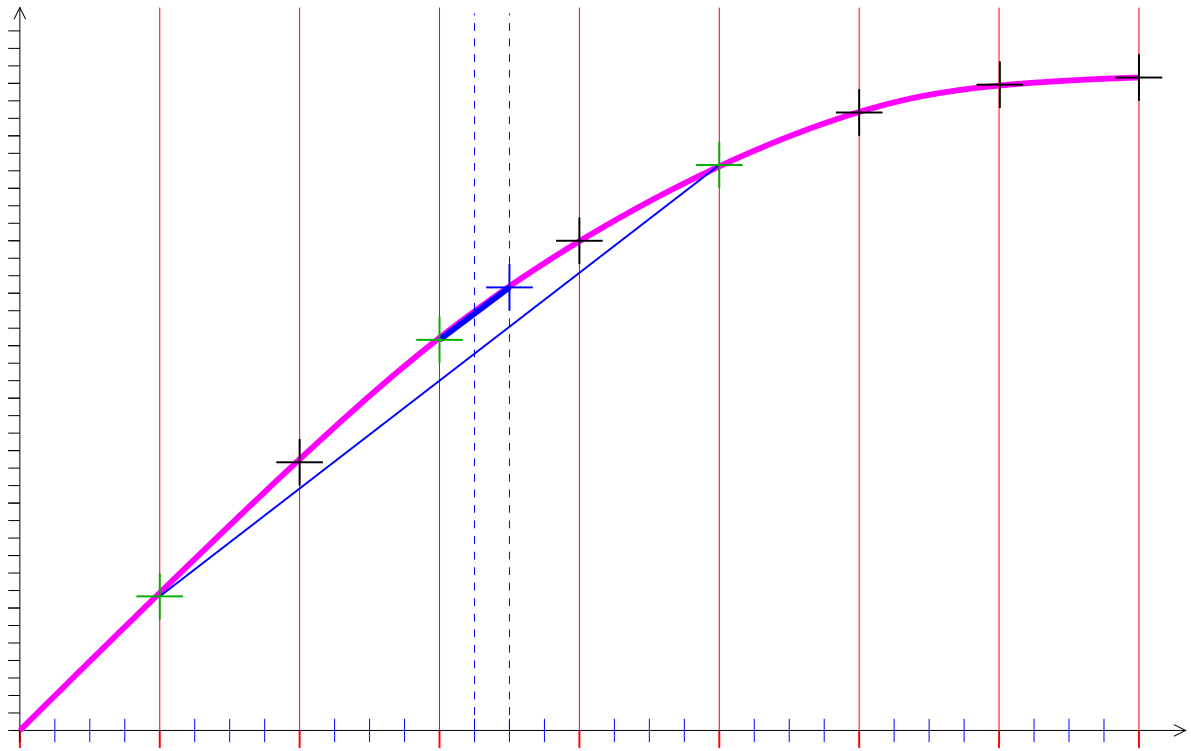
$$Bf'(A) \approx f(A + B) - f(A)$$

Finalement,

$$f(A + 2^{-\beta}B) \;\; \approx \;\; f(A) \; + \; 2^{-\beta} \left(\; f(A + B) - f(A) \; \right)$$

Méthodes ATA au second ordre

Une différence centrée permet d'évaluer la dérivée avec une précision bien meilleure (terme d'erreur du troisième ordre) :



Méthodes ATA au second ordre

On utilise

$$Bf'(A) \approx \frac{f(A+B) - f(A-B)}{2}$$

Il faut alors

- calculer (en parallèle) $A+B$ et $A-B$,
- lire les valeurs $f(A)$, $f(A+B)$, $f(A-B)$,
- reconstruire

$$f(A + 2^{-\beta}B) \approx f(A) + 2^{-\beta-1}\left(f(A+B) - f(A-B) \right)$$

Méthodes ATA vraiment au second ordre

Mais avec ces trois lectures de tables on peut même évaluer un terme de second ordre :

$$f(A + 2^{-\beta}B) \approx f(A) \; + \; 2^{-\beta}Bf'(A) \; + \; \frac{(2^{-\beta}B)^2}{2} f''(A)$$

Le terme en $f''(A)$ est calculé par

$$\begin{aligned} f''(A) &\approx \frac{f'(A + B/2) - f'(A - B/2)}{B} \\ &\approx \frac{\frac{f(A+B)-f(A)}{B} - \frac{f(A)-f(A-B)}{B}}{B} \\ &\approx \frac{f(A + B) - 2f(A) + f(A - B)}{B^2} \end{aligned}$$

Finalement

$$
\begin{aligned}
f(A + 2^{-\beta} B) \quad \approx \quad & f(A) \\
& + 2^{-\beta-1} \left(\; f(A+B) - f(A-B) \; \right) \\
& + 2^{-2\beta} \left(\; f(A+B) - 2f(A) + f(A-B) \; \right)
\end{aligned}
$$

Précision de ces approximations

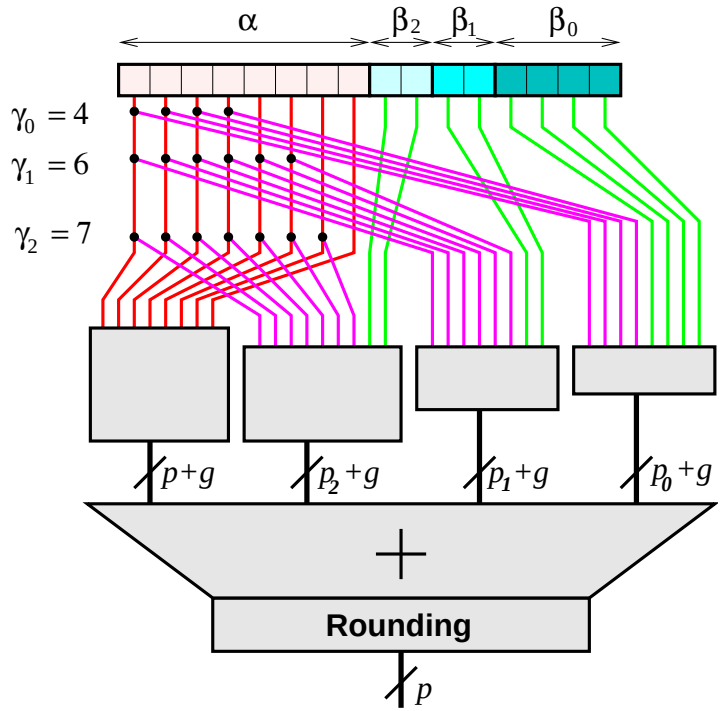
On peut mesurer l'erreur de méthode (qui dépend de f, α et β).
On constate (sans surprise) que

- la précision est d'autant plus grande que β est petit devant α ;
- le terme de correction de second ordre est inutile dès que $\beta > \alpha/2$.

La déception c'est que c'est toujours plus mauvais que la méthode multipartite...

Tas de bits (Hassler et Takagi)

Pas fait exprès



Or que constate-t-on à la fin du papier d'Hassler et Takagi ?

H. Hassler and N. Takagi. Function evaluation by table look-up and addition, in *12th IEEE Symposium on Computer Arithmetic*, pages 10–16, Bath, UK, 1995. IEEE Computer Society Press.

Sauf que pour en arriver là,

ils ne font pas du tout comme nous...

! ? ! ! ?

(cela dit on est meilleurs, c'est l'essentiel)

Principe de la méthode d'Hassler et Takagi

- On approxime $f(x)$ par un polynôme[1] : $f(x) \approx \sum_{i=0}^{d} a_i x^i$
- On écrit x et les a_i en binaire[2] : $x = \sum_{j \in J} 2^j x_j, \quad a_i = \sum_k 2^k a_{ik}$
- On développe le polynôme (les x^i), éventuellement en simplifiant :
  - en virant les termes où intervient un a_{ik} nul,
  - en simplifiant par $x_j x_j = x_j$
  - en simplifiant par $P + P = 2P$

Finalement : $f(x) \approx \sum_p 2^l \sum_q P_{pq}$ avec $P_{pq} = \prod_{j \in I_{pq} \subset J} x_j$ et $I_{pq} \subset J$

[1] Eux ils disent un DL.

[2] Voui, y faut qu'ils soient tous positifs...

D'une grosse somme à une approximation multipartite à m tables :

- soit les m ensembles d'indices de x : I_1, ..., I_m (les entrées des tables) ;
- il suffit de regrouper nos P_{pq} en $m+1$ ensembles :
  - ceux qui n'ont que des indices de I_1, qu'on va entasser dans une table T_1,
  - ...
  - ceux qui n'ont que des indices de I_m, qu'on va entasser dans une table T_m,
  - les autres, qui représentent l'erreur de méthode, dont on appelle donc la somme E.
- Bref $f(x) \approx T_1(x_{/I_1}) + ... + T_m(x_{/I_m}) \ + \ E(x)$
- Problème : le choix des I_i minimisant E et les tailles de tables. H&T ont une heuristique assez heuristique.
- En ce qui concerne l'erreur de méthode [3] : comme tous nos termes sont positifs,
  E(x) est positif, et maximal pour $x = 11...11_b$

[3] Personne n'a encore parlé d'erreur d'arrondi

Plus intéressant : récapituler dans l'autre sens.
Si P est un polynôme à termes tous positifs,

$$\begin{aligned} P(x) &= P(x_{/I_1}) \\ &+ P(x_{/I_2}) - P(x_{/I_1 \cap I_2}) \\ &+ P(x_{/I_3}) - P(x_{/I_1 \cap I_3}) - P(x_{/I_2 \cap I_3}) + P(x_{/I_1 \cap I_2 \cap I_3}) \\ &+ \ldots \\ &+ E(x) \end{aligned}$$

avec $E(x)$ positif, et maximal pour $x = 11...11_b$.

Eh ben j'ai mis trois jours à comprendre cela dans le papier d'H&T...

Ce qui est magnifique, c'est qu'on peut prendre un polynôme de degré arbitraire.
H&T ne parlent que de DLs et j'ai un peu l'impression qu'ils passent à la limite en écrivant f à la place de P.

D'où la méthode pour construire une archi pour évaluer f à la précision ϵ :

1. approximer f par un polynôme à coeffs tous positifs (erreur $\epsilon_1 << \epsilon$).
2. si (comme) ce n'est pas possible, écrire $f \approx P_{\oplus} - P_{\ominus}$
3. pour chaque polynôme, trouver une bonne décomposition en tables (erreur $\epsilon_{2\oplus}$ et $\epsilon_{2\ominus}$).
4. gérer les erreurs d'arrondi dans les tables produites comme en multipartite.

Pour les tables multipartites une exploration exhaustive était possible. Ici ce n'est plus le cas au dela de 8 bits...

Et pourquoi qu'on obtient-y une architecture patapartite ?

- Bipartite :

$$\begin{aligned} P(x) &= P(x_{/I_1}) \\ &+ P(x_{/I_2}) - P(x_{/I_1 \cap I_2}) \\ &+ E(x) \end{aligned}$$

- Multipartite
  Euh... c'est moins clair
- on reste dans les approximations linéaires ou pas ?

- Quels sont les polynômes qui donnent de bonnes approx par cette méthode ?
  - Supposons un polynôme de degré d
  - On a éparpillé des termes sur un intervalle de puissances de 2 de taille :
    $d+1 \times$ la précision de x.
  - On veut en sortie plus ou moins la précision de x.
  - Les bits de poids fort de x sont partout, d'où les I_i qu'on obtient.
  - Dans l'heuristique de partitionnement, on peut laisser dans E sans remord des termes d'ordre $2^{-\text{beaucoup}}$... sauf s'ils sont trop nombreux.
  - Pour le reste on peut surement améliorer l'heuristique de H&T (en commencant par ne pas imposer les tables toutes de même taille) mais vaut-ce le coup ?
  - Une citation de H&T :
    Especially important is the convergence rate. A function that converges quickly has in the significant bit range relatively small bit products, which are easier to contain in small tables.

Finalement

- Je suis pas certain que cela serve
- La méthode est très puissante, mais on a perdu les propriétés de haut niveau de la fonction (continuité) qui rendaient l'analyse multipartite exhaustive possible
- On pourrait regrouper les termes en petits multiplieurs, etc
- Pourquoi on n'a pas tout fait en complément à 2 ? Dans nos multiplieurs on avait des tas de bits pondérés positifs et négatifs.
  - Pasque on perd la borne sur l'erreur facile à calculer.
  - Il y a peut-être moyen de la récupérer...

Méthodes polynômiales et rationnelles

Principe général

- On découpe l'intervalle d'entrée en morceaux.
- Sur chaque morceau on utilise une approximation polynômiale ou rationnelle.

Coût :

- Rappel : évaluation d'un polynôme de degré d par la méthode de Horner :
  d additions et d multiplications.
- matériel : des multiplieurs, des additionneurs, et des tables pour stocker les coefficients.
- temps : quelques multiplications et additions.

Compromis (pour une précision fixée) :

- petit degré = rapide mais beaucoup d'intervalles donc de grosses tables,
- gros degré = plus lent mais tables plus petites.

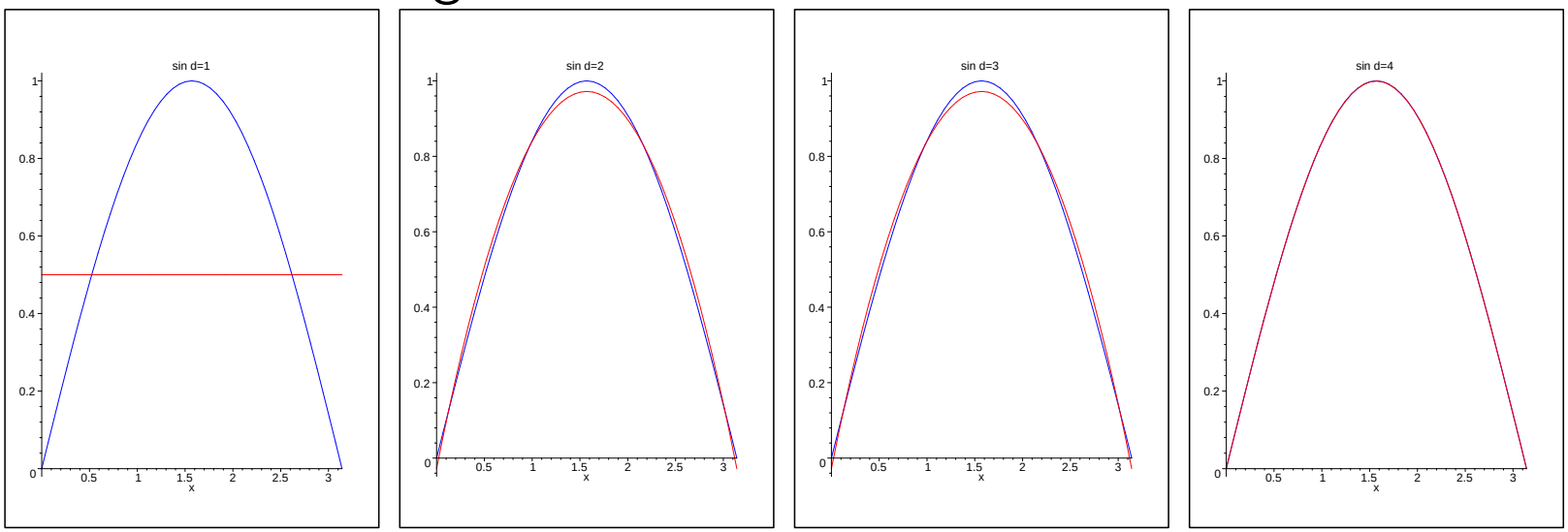
Erreur d'approx

Précision théorique (en bits) pour sin (polynômes[4] minimax, degré d) :

d	$[0, \pi/4]$	$[0, \pi/2]$	$[0, \pi]$	$[0, 2\pi]$
1	6	3	0	0
2	8	6	5	0
3	14	9	5	3
4	17	13	10	3
5	23	17	10	7

[4] Évaluation d'un polynôme de degré d par Horner $\Longrightarrow$ $d+$ et $d\times$

Exemple : approximations polynômiales de sinus entre 0 et π en fonction du degré :



Voyons à présent quelques variations intelligentes.

L'approche Piñeiro et Bruguera

- Polynôme du second degré
- Un multiplieur et un "metteur au carré", puis un arbre d'addition
- Ils sont (pas tellement) meilleurs que les multipartites pour 24 bits, pas avant

Dans leur papier (Arith 15) ils ne parlent que de fonctions x^n mais cela doit marcher pour toutes les fonctions.

L'approche Liddicoat et Flynn

- développement en série de $1/x$
- plein d'unités d'élévation à la puissance, puis un arbre d'addition

L'approche Muller-Defour

On en déduit, en ajoutant :

$$\varphi = \frac{1}{2} x_2^2 2^{-4k} f''(x_0)$$
$$\& \quad \psi = -\frac{1}{6} x_2^3 2^{-4k} f'''(x_0) \quad \Big\} \text{ même table, adressée par } x_0 \text{ et } x_2$$

(qui vaut $-\frac{1}{6} x_2^3 2^{-4k} f'''(x_0 + x_1 2^{-k}) + \varepsilon\left(\frac{1}{6} 2^{-5k} f^{(4)}\right)$)

$$f(x) = f(x_0 + x_1 2^{-k}) + \frac{1}{2}(\alpha - \beta) 2^{-k} + \frac{1}{2}(\gamma - \delta) 2^{-2k}$$
$$+ (\varphi + \psi)(x_1 2^{-k}) + \varepsilon\left(\frac{1}{120} 2^{-6k} f^{(5)}\right) + \varepsilon\left(\frac{1}{120} 2^{-7k} f^{(5)}\right)$$
$$+ \varepsilon\left(\frac{1}{2} 2^{-5k} f^{(3)}\right) + \varepsilon\left(2^{-5k} f^{(2)}\right) + \varepsilon\left(\frac{1}{6} 2^{-6k} f^{(3)}\right)$$
$$+ \varepsilon\left(\frac{1}{6} 2^{-5k} f^{(4)}\right)$$

Si $k \geq 5$ alors $2^{-k} \leq \frac{1}{32}$

et le terme d'erreur devient

$$\varepsilon\left(\frac{11}{1280} 2^{-6k} f^{(5)}\right) + \varepsilon\left(\frac{92}{192} 2^{-5k} f^{(3)}\right)$$
$$+ \varepsilon\left(2^{-5k} f^{(2)}\right) + \varepsilon\left(\frac{1}{6} 2^{-5k} f^{(4)}\right)$$

et si toutes les dérivées sont majorées par M
(cos de l'exp, des fctions trigos ...) l'erreur devient

$$\frac{63691}{2^{16}} M 2^{-5k} \leq 1.68 \cdot 2^{-5k} M$$

- $f(x)$ avec $x \in [0,1[$ en virgule fixe sur $n = 4k + p$ bits
- donc $x = x_0 + x_1 2^{-k} + x_2 2^{-2k} + x_3 2^{-3k} + x_4 2^{-4k}$,
- et les x_i sont des nombres codés sur k bits appartenant à $[0,1[$, sauf pour x_4 qui lui est sur p bits, où $p < k$.
- et on fait un Taylor d'ordre 5
- et on le fait développer par Maple (Defour est pas fou)
- et on garde seulement les termes utiles pour le résultat final (précision visée est de l'ordre de 2^{-4k-p})

Bref on part de

$$
\begin{array}{rcll}
f(x) & = & f(x_0) & (T_0) \\
 & + & [x-x_0]f'(x_0) & (T_1) \\
 & + & \frac{1}{2}[x-x_0]^2 f''(x_0) & (T_2) \\
 & + & \frac{1}{6}[x-x_0]^3 f'''(x_0) & (T_3) \\
 & + & \frac{1}{24}[x-x_0]^4 f^{(4)}(x_0) & (T_4) \\
 & + & \frac{1}{120}[x-x_0]^5 f^{(5)}(x_0) & (T_5) \\
 & + & \epsilon_1 &
\end{array}
$$

où

$$\epsilon_1 = \frac{1}{720}([x_1 2^{-k} + x_2 2^{-2k} + x_3 2^{-3k} + x_4 2^{-4k}]^6) f^{(6)} < \frac{1}{720} 2^{-6k} \max |f^{(6)}|$$

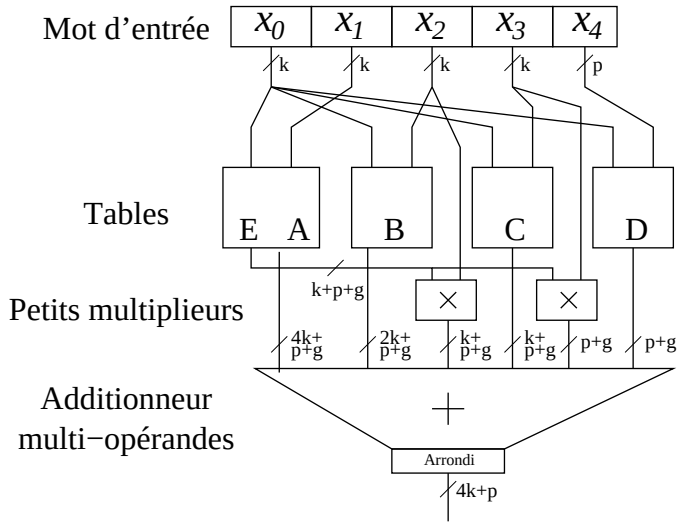
et on arrive à

$$\begin{aligned} f(x) &= A(x_0,x_1)+B(x_0,x_2)+C(x_0,x_3)+D(x_0,x_4)+ \\ & \quad x_2\times E(x_0,x_1)+x_3 2^{-k}\times E(x_0,x_1)+\epsilon_f \end{aligned}$$

où

$$\begin{aligned} A(x_0,x_1) &= f(x_0)+x_1 2^{-k} f'(x_0)+\tfrac{1}{2}x_1^2 2^{-2k} f''(x_0)+ \\ & \quad \tfrac{1}{6}x_1^3 2^{-3k} f'''(x_0)+\tfrac{1}{24}x_1^4 2^{-4k} f^{(4)}(x_0)+ \\ & \quad \tfrac{1}{120}x_1^5 2^{-5k} f^{(5)}(x_0)+x_1(1/2)2^{-5k} f''(x_0)+ \\ B(x_0,x_2) &= x_2 2^{-2k} f'(x_0)+\tfrac{1}{2}x_2^2 2^{-4k} f''(x_0)+(1/2)x_2 2^{-5k} f''(x_0) \\ C(x_0,x_3) &= x_3 2^{-3k} f'(x_0) \\ D(x_0,x_4) &= x_4 2^{-4k} f'(x_0) \\ E(x_0,x_1) &= x_1 2^{-3k} f''(x_0)+\tfrac{1}{2}x_1^2 2^{-4k} f'''(x_0)+ \\ & \quad \tfrac{1}{6}x_1^3 2^{-5k} f^{(4)}(x_0)+\tfrac{1}{2}x_1(1/2)2^{-5k} f'''(x_0) \\ \epsilon_f &\leq \epsilon_1+\epsilon_2+\epsilon_3+\epsilon_4+\epsilon_5 \\ &\leq \tfrac{1}{720}2^{-6k}\max|f^{(6)}|+\tfrac{1}{2}2^{-5k}\max|f''|+\tfrac{1}{3}2^{-6k}\max|f'''|+ \\ & \quad \tfrac{1}{24}2^{-6k}\max|f^{(4)}|+\tfrac{1}{120}2^{-6k}\max|f^{(5)}| \\ &\leq 2^{-5k}\left[\tfrac{1}{2}\max|f''|+\tfrac{1}{3}\max|f'''|+\right. \\ & \quad \left.\tfrac{1}{24}\max|f^{(4)}|+\tfrac{1}{120}\max|f^{(5)}|+\tfrac{1}{720}\max|f^{(6)}|\right] \end{aligned}$$

Architecture



- 2 petites multiplications et 5 additions
- Les dérivées successives doivent décroître suffisamment rapidement pour rendre les termes absents des tables non significatifs par rapport aux autres.
- bref cela ne marche pas pour toutes les fonctions
- et pas pour toutes les tailles de bits.

Résultats

f	k	p	Taille de table	Nombre de bits corrects	Taille de référence
sin $[0, \pi/4[$	3	2	2768	14	3712
	4	3	15040	19	29440
	5	3	70528	23	138624
exp $[0, 1[$	3	2	3232	14	6272
	4	3	16256	19	56320
	5	4	82432	24	366080
$2^x - 1$ $[0, 1[$	3	2	3392	14	7168
	4	3	18048	19	56320
	5	4	89600	24	259584

C'est un peu rigide tout cela

- 2 petites multiplications et 5 additions
- bref cela ne marche pas pour toutes les fonctions
- et pas pour toutes les tailles de bits.

Jérémie l'a dérigidifié (FPL 2004) et généralisé à n'importe quel degré (ASSAP 2005).

Le retour du fils de la vengeance de Jérémie