

Produit scalaire dans $\mathbb{Z}/p\mathbb{Z}$ en arithmétique flottante

Jérémy JEAN

LIP6/PEQUAN - Université Pierre et Marie Curie (Paris 6)

RAIM 2009

ENS Lyon, 27 Octobre 2009



- Motivations
- Rappels
 - Arithmétique flottante
 - Corps finis
- Calcul du produit scalaire
 - Première méthode
 - Deuxième méthode
- Comparaison
- Bilan et perspectives

Motivations :

- Produit scalaire : fonction de base du calcul matriciel
- Algorithmes rapides pour le calcul scientifique
- Cryptologie
- Codes correcteurs
- Calcul formel

Nombres flottants

Nombres flottants normalisés $\mathbb{F} \subset \mathbb{R}$:

$$x = \pm \underbrace{x_0.x_1 \dots x_{M-1}}_{\text{mantissa}} \times b^e, \quad 0 \leq x_i \leq b-1, \quad x_0 \neq 0$$

b : base, M : précision, e : exposant t.q. $e_{\min} \leq e \leq e_{\max}$

Approximation de \mathbb{R} par \mathbb{F} , en arrondissant avec $\mathbf{fl} : \mathbb{R} \rightarrow \mathbb{F}$.

Soit $x \in \mathbb{R}$. Alors :

$$\mathbf{fl}(x) = x(1 + \delta), \quad |\delta| \leq \mathbf{u}$$

Unité d'arrondi $\mathbf{u} = b^{1-M}$ pour l'arrondi vers zéro

Standard pour l'arithmétique flottante

Soit $x, y \in \mathbb{F}$ et $\circ \in \{+, -, \cdot, /\}$.

Le résultat flottant $\mathbf{fl}(x \circ y)$ n'est en général pas représentable en flottant :

$$\mathbf{fl}(x \circ y) = (x \circ y)(1 + \delta), \quad |\delta| \leq \mathbf{u}$$

Standard IEEE 754 (1985)

Type	Taille	Mantisse	Exposant	Unité d'arrondi	Interval
Simple	32 bits	23+1 bits	8 bits	$\mathbf{u} = 2^{1-24} \approx 1,92 \times 10^{-7}$	$\approx 10^{\pm 38}$
Double	64 bits	52+1 bits	11 bits	$\mathbf{u} = 2^{1-53} \approx 2,22 \times 10^{-16}$	$\approx 10^{\pm 308}$

Corps finis premiers $\mathbb{Z}/p\mathbb{Z}$ (p premier)

$\mathbb{Z}/p\mathbb{Z} = \{0, 1, \dots, p - 1\}$ est un corps fini de caractéristique p

Corps finis premiers $\mathbb{Z}/p\mathbb{Z}$ (p premier)

$\mathbb{Z}/p\mathbb{Z} = \{0, 1, \dots, p-1\}$ est un corps fini de caractéristique p

Opérations dans le corps, pour $a, b \in \mathbb{Z}/p\mathbb{Z}$:

- Somme : $a + b \in \{0, \dots, 2(p-1)\} \rightarrow a + b \pmod{p} \in \mathbb{Z}/p\mathbb{Z}$
- Produit : $ab \in \{0, \dots, (p-1)^2\} \rightarrow ab \pmod{p} \in \mathbb{Z}/p\mathbb{Z}$

Corps finis premiers $\mathbb{Z}/p\mathbb{Z}$ (p premier)

$\mathbb{Z}/p\mathbb{Z} = \{0, 1, \dots, p-1\}$ est un corps fini de caractéristique p

Opérations dans le corps, pour $a, b \in \mathbb{Z}/p\mathbb{Z}$:

- Somme : $a + b \in \{0, \dots, 2(p-1)\} \rightarrow a + b \pmod{p} \in \mathbb{Z}/p\mathbb{Z}$
- Produit : $ab \in \{0, \dots, (p-1)^2\} \rightarrow ab \pmod{p} \in \mathbb{Z}/p\mathbb{Z}$

Réduction modulo p pour $a \in \mathbb{Z}/p\mathbb{Z}$:

$$a \pmod{p} = a - \left\lfloor \frac{a}{p} \right\rfloor p = a - \lfloor a \cdot \text{inv}P \rfloor p$$

But et hypothèses

Soit $p \geq 3$ un nombre premier, et $(a_i)_i, (b_i)_i$ deux vecteurs de N scalaires de $\mathbb{Z}/p\mathbb{Z}$. On cherche à calculer le produit scalaire de a et b dans $\mathbb{Z}/p\mathbb{Z}$:

$$a \cdot b = \sum_{i=1}^N a_i b_i \pmod{p}$$

But et hypothèses

Soit $p \geq 3$ un nombre premier, et $(a_i)_i, (b_i)_i$ deux vecteurs de N scalaires de $\mathbb{Z}/p\mathbb{Z}$. On cherche à calculer le produit scalaire de a et b dans $\mathbb{Z}/p\mathbb{Z}$:

$$a \cdot b = \sum_{i=1}^N a_i b_i \pmod{p}$$

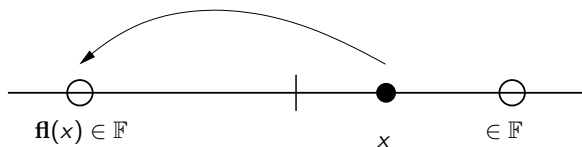
Hypothèses :

- Les entiers sont codés dans des flottants $\rightarrow \mathbb{F} \cap \mathbb{N}$
- On limite p à : $p - 1 < 2^{M-1}$
- Tous les nombres mis en jeu sont supposés **positifs**
- On se place dans le mode d'**arrondi vers zéro**

Arrondi vers zéro sur \mathbb{R}^+

Soit $x \in \mathbb{R}^+$ $\mathbf{fl}(x)$ est l'arrondi de x dans \mathbb{F}

- Equivalent à une troncature

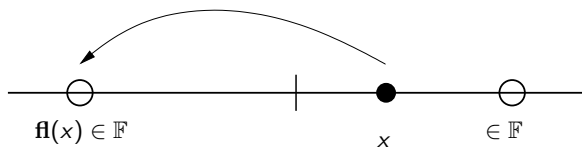


Arrondi vers zéro sur \mathbb{R}^+

Soit $x \in \mathbb{R}^+$ $\mathbf{fl}(x)$ est l'arrondi de x dans \mathbb{F}

- Equivalent à une troncature
- Le résultat flottant approché est inférieur au résultat exact :

$$\forall x \in \mathbb{R}^+, \mathbf{fl}(x) \leq x$$



Arrondi vers zéro sur \mathbb{R}^+

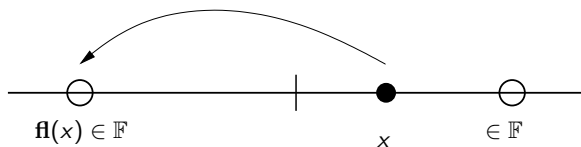
Soit $x \in \mathbb{R}^+$ $\mathbf{fl}(x)$ est l'arrondi de x dans \mathbb{F}

- Equivalent à une troncature
- Le résultat flottant approché est inférieur au résultat exact :

$$\forall x \in \mathbb{R}^+, \mathbf{fl}(x) \leq x$$

- L'erreur d'arrondi est positive :

$$\forall x \in \mathbb{R}^+, x - \mathbf{fl}(x) \geq 0$$



Problème : le résultat d'une opération flottante peut ne pas être représentable en flottant.

Solution : Transformations exactes (*Error-free transformations*) :

- Somme non-évaluée de deux flottants :
 - le résultat flottant de l'opération
 - l'erreur commise (représentable dans \mathbb{F} dans notre cas)
- Pour $a, b \in \mathbb{F} \cap \mathbb{N}$ et $\circ \in \{+, *\}$,

$$a \circ b = \mathbf{fl}(a \circ b) + e, \text{ avec } e \in \mathbb{F},$$

exactement mathématiquement.

Transformation exacte pour le produit (1/2)

Pour $a, b, c \in \mathbb{F}$,

- $\text{FMA}(a, b, c)$ est l'arrondi vers zéro de $a \cdot b + c \in \mathbb{F}$

Algorithme 1 (EFT pour le produit de deux flottants)

```
fonction  $[x, y] = \text{TwoProductFMA}(a, b)$ 
```

```
   $x = \text{fl}(a \cdot b)$ 
```

```
   $y = \text{FMA}(a, b, -x)$ 
```

Le FMA est dans la norme IEEE 754 depuis Août 2008

Transformation exacte pour le produit (2/2)

Théoreme 1

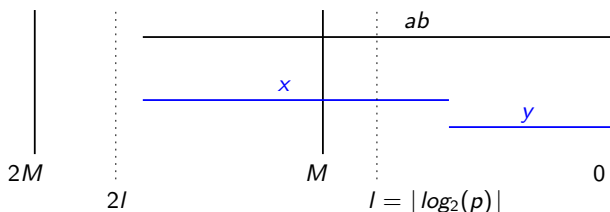
Soient $a, b \in \mathbb{F} \cap \mathbb{N}$ et soient $x, y \in \mathbb{F}$ tels que

$$[x, y] \leftarrow \text{TwoProductFMA}(a, b)$$

Alors :

$$ab = x + y, \quad x = \mathbf{fl}(ab), \quad 0 \leq y < \mathbf{u.ufp}(x), \quad 0 \leq x \leq ab$$

L'algorithme `TwoProductFMA` nécessite 2 flops.



Division euclidienne binaire (1/2)

Pour $a, d \in \mathbb{F} \cap \mathbb{N}$, $d \neq 0$, la division euclidienne de a par d s'écrit :

$$a = qd + r, \quad 0 \leq r < d$$

Pour $a \in \mathbb{F} \cap \mathbb{N}$ et $\sigma = 2^k$, $\sigma \geq a$, on définit :

Algorithme 2 (Séparation d'un flottant en deux)

```
fonction  $[x, y] = \text{ExtractScalar}(\sigma, a)$ 
```

$$q = \mathbf{fl}(\sigma + a)$$

$$x = \mathbf{fl}(q - \sigma)$$

$$y = \mathbf{fl}(x - a)$$

Rappel : \mathbf{fl} arrondi vers zéro

Algorithme initialement proposé par S. Rump.

Division euclidienne binaire (2/2)

Théoreme 2

Soient $a \in \mathbb{F} \cap \mathbb{N}$, $\sigma = 2^k$, $\sigma \geq a$ et soient $x, y \in \mathbb{F}$ tels que

$$[x, y] \leftarrow \text{ExtractScalar}(\sigma, a)$$

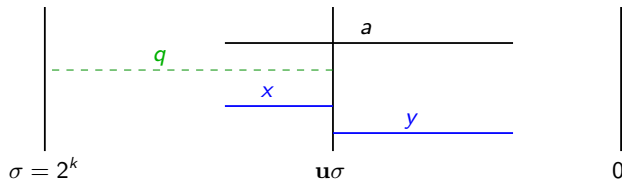
Alors :

$$a = x + y, \quad 0 \leq y < u\sigma, \quad 0 \leq x \leq a, \quad x \in u\sigma\mathbb{N}$$

L'algorithme `ExtractScalar` nécessite 3 flops.

Remarque :

$$a = x + y = x' u\sigma + r, \quad x' \in \mathbb{N}, \quad 0 \leq r < u\sigma$$



$$a \cdot b = \sum_{i=1}^N a_i b_i \pmod{p}$$

Deux approches légèrement différentes

Calcul du produit scalaire

$$a \cdot b = \sum_{i=1}^N a_i b_i \pmod{p}$$

Deux approches légèrement différentes

- Première méthode :

$$\lambda(p-1) < 2^{M-1} \quad \text{avec} \quad \lambda \in \mathbb{N}^*$$

Calcul du produit scalaire

$$a \cdot b = \sum_{i=1}^N a_i b_i \pmod{p}$$

Deux approches légèrement différentes

- Première méthode :

$$\lambda(p-1) < 2^{M-1} \quad \text{avec} \quad \lambda \in \mathbb{N}^*$$

- Deuxième méthode :

$$p-1 < 2^{M-1} \quad \text{mais} \quad N < 2^{M/2}$$

En **double**, la taille maximum des vecteurs est donc $2^{53/2} \approx 10^8$.

Première méthode

Calcul du produit scalaire : première méthode

Hypothèse : $\lambda(p - 1) < 2^{M-1}$

Conséquences :

- La somme de λ entiers du corps tient dans une mantisse
- On peut délayer les réductions modulo p tous les λ

Calcul du produit scalaire : première méthode

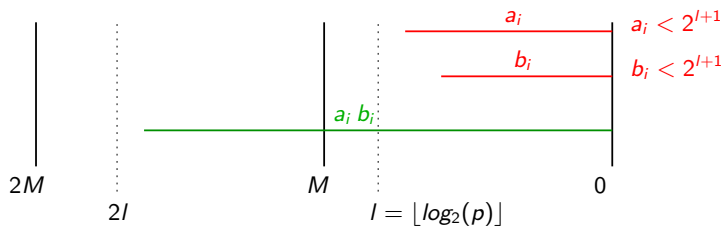
Hypothèse : $\lambda(p-1) < 2^{M-1}$

Conséquences :

- La somme de λ entiers du corps tient dans une mantisse
- On peut délayer les réductions modulo p tous les λ

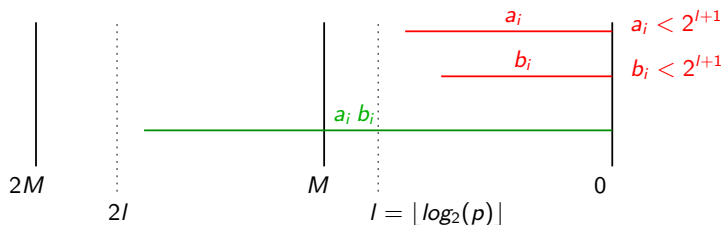
Jean-Guillaume Dumas : $\lambda(p-1)^2 < 2^M$

Première méthode : principe



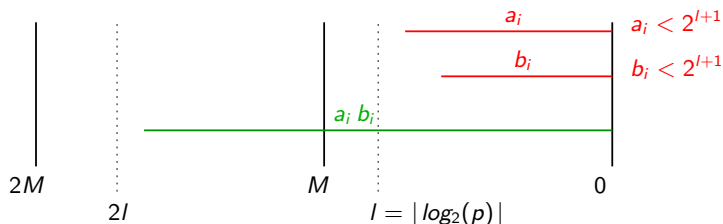
Soit $l = \lfloor \log_2(p) \rfloor$

Première méthode : principe



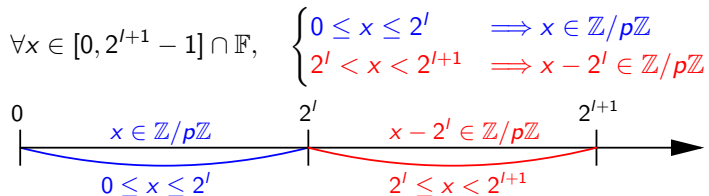
Soit $l = \lfloor \log_2(p) \rfloor \implies \mathbf{ufp}(p) := 2^l \neq p$ ($\mathbf{ufp}(p)$ = bit de poids fort de p)

Première méthode : principe



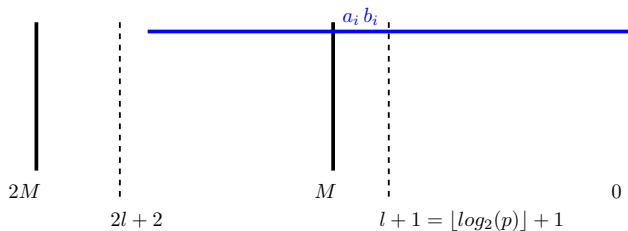
Soit $l = \lfloor \log_2(p) \rfloor \implies \mathbf{ufp}(p) := 2^l \neq p$ ($\mathbf{ufp}(p)$ = bit de poids fort de p)

- $p \geq 3$ et premier donc : $\mathbf{ufp}(p) < p < 2 \cdot \mathbf{ufp}(p)$ i.e. $2^l < p < 2^{l+1}$
- Constatation :



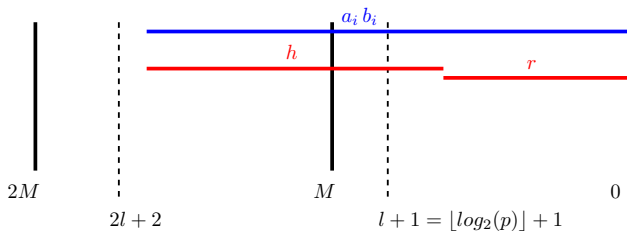
Première méthode : principe

$$\text{TwoProductFMA} \implies a_i b_i = h + r$$



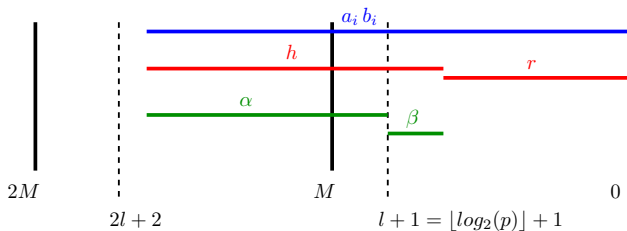
Première méthode : principe

$$\text{TwoProductFMA} \implies a_i b_i = h + r$$



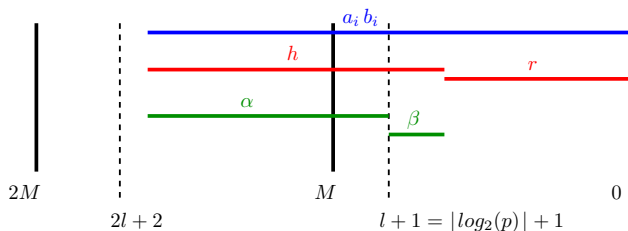
Première méthode : principe

$$\text{TwoProductFMA} \implies a_i b_i = h + r$$



Première méthode : principe

$$\text{TwoProductFMA} \implies a_i b_i = h + r$$

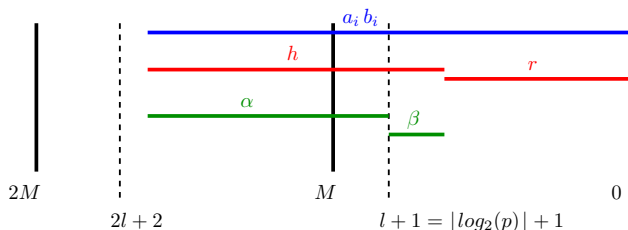


Après découpe avec **ExtractScalar** :

- $h = \alpha + \beta$ avec $0 \leq \alpha/2^{l+1}, \beta < 2^{l+1}$

Première méthode : principe

$$\text{TwoProductFMA} \implies a_i b_i = h + r$$

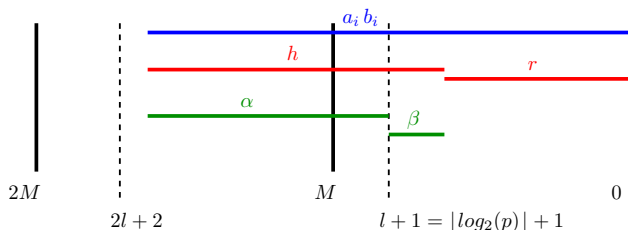


Après découpe avec **ExtractScalar** :

- $h = \alpha + \beta$ avec $0 \leq \alpha/2^{l+1}, \beta < 2^{l+1}$
- On accumule $\alpha/2^{l+1} \in \mathbb{Z}/p\mathbb{Z}$ ou $\alpha/2^{l+1} - 2^l \in \mathbb{Z}/p\mathbb{Z}$
- On compte le nombre n_α de -2^l ajoutés

Première méthode : principe

$$\text{TwoProductFMA} \implies a_i b_i = h + r$$



Après découpe avec **ExtractScalar** :

- $h = \alpha + \beta$ avec $0 \leq \alpha/2^{l+1}, \beta < 2^{l+1}$
- On accumule $\alpha/2^{l+1} \in \mathbb{Z}/p\mathbb{Z}$ ou $\alpha/2^{l+1} - 2^l \in \mathbb{Z}/p\mathbb{Z}$
- On compte le nombre n_α de -2^l ajoutés
- Idem pour β : $\beta \in \mathbb{Z}/p\mathbb{Z}$ ou $\beta - 2^l \in \mathbb{Z}/p\mathbb{Z}$
- $n_\beta :=$ nombre de correction de -2^l pour β

Première méthode : calcul final

$$\begin{aligned} a \cdot b &= \sum_{i=1}^N a_i b_i \\ &= \sum_{i=1}^N \alpha_i + \sum_{i=1}^N \beta_i + \sum_{i=1}^N r_i \\ &= \sum_{n_\alpha} (\alpha_i / 2^{l+1} - 2^l) + \sum_{N-n_\alpha} \alpha_i / 2^{l+1} + \sum_{n_\beta} (\beta_i - 2^l) + \sum_{N-n_\beta} \beta_i + \sum_N r_i \\ &\quad + (n_\alpha + n_\beta) 2^l \end{aligned}$$

Première méthode : calcul final

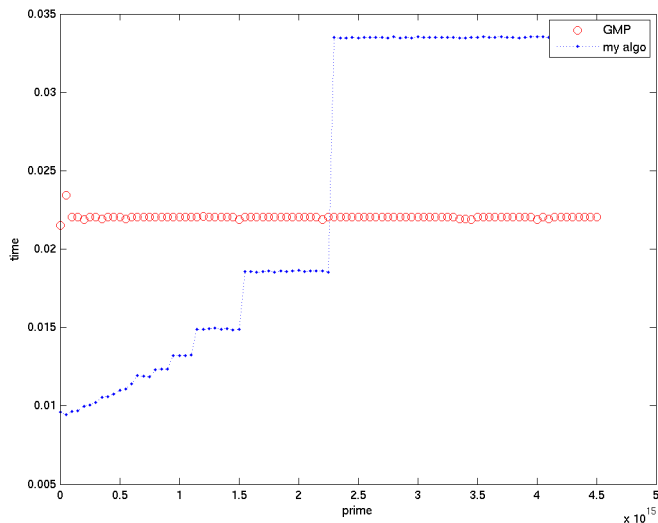
$$\begin{aligned} a \cdot b &= \sum_{i=1}^N a_i b_i \\ &= \sum_{i=1}^N \alpha_i + \sum_{i=1}^N \beta_i + \sum_{i=1}^N r_i \\ &= \sum_{n_\alpha} (\alpha_i / 2^{l+1} - 2^l) + \sum_{N-n_\alpha} \alpha_i / 2^{l+1} + \sum_{n_\beta} (\beta_i - 2^l) + \sum_{N-n_\beta} \beta_i + \sum_N r_i \\ &\quad + (n_\alpha + n_\beta) 2^l \end{aligned}$$

$\lambda(p-1) < 2^{M-1} \implies$ sommation par paquets de λ puis réduction mod p

- Sur Itanium2
- Avec FMA
- En double précision ($p - 1 < 2^{53-1}$)
- Comparaison avec GMP

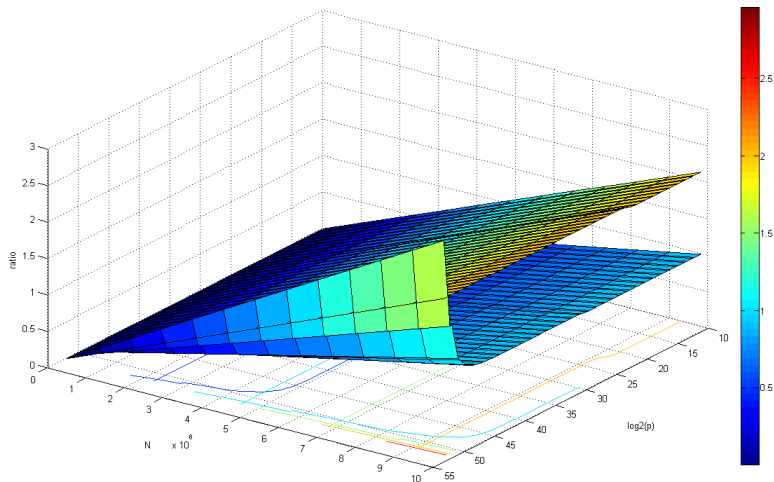
Première méthode : Performances sur Itanium2 (1/4)

FIG.: Comparaison avec GMP : temps= $f(p \in [2^{23}, 2^{52}])$, pour $N = 10^5$



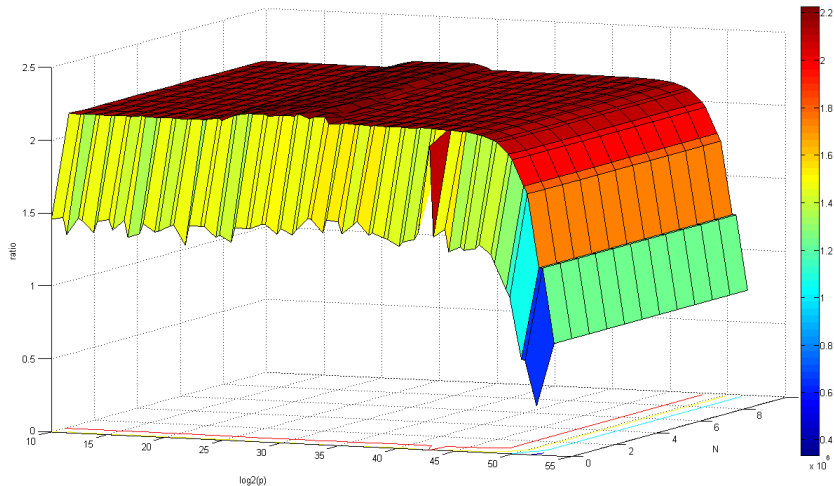
Première méthode : Performances sur Itanium2 (2/4)

FIG.: Comparaison avec GMP : temps= $f(N, \log_2(p))$ — GMP en haut



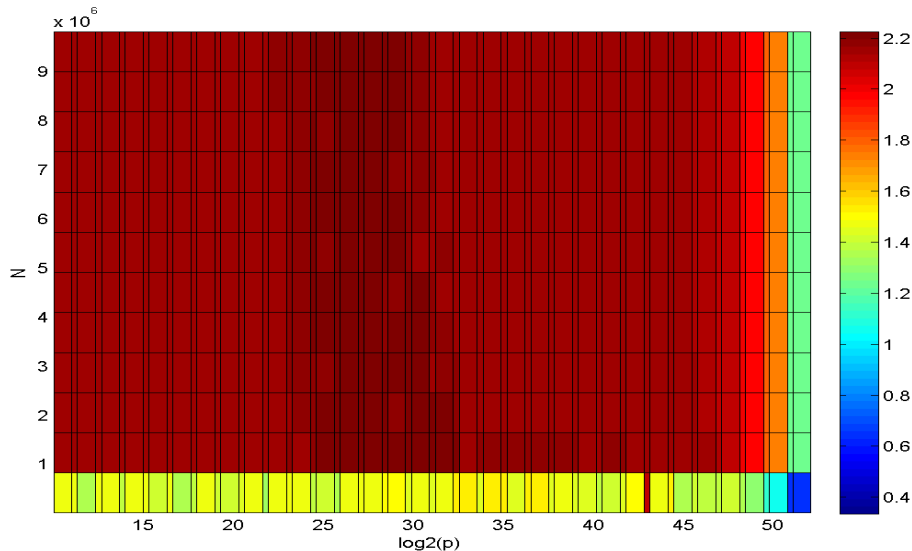
Première méthode : Performances sur Itanium2 (3/4)

FIG.: Surface : $\text{ratio} = \text{temps}(GMP) / \text{temps}(\text{algo}) = f(N, \log_2(p))$



Première méthode : Performances sur Itanium2 (4/4)

FIG.: $\text{ratio} = \text{temps}(GMP) / \text{temps}(\text{algo}) = f(N, \log_2(p))$



Deuxième méthode

Calcul du produit scalaire : deuxième méthode

Hypothèse : $p - 1 < 2^{M-1}$ et $N < 2^{M/2}$

Calcul du produit scalaire : deuxième méthode

Hypothèse : $p - 1 < 2^{M-1}$ et $N < 2^{M/2}$

Idée :

- Se ramener à des entiers représentables sur au plus une demi-mantisse
- Les sommer ensemble de manière exacte
- Réduire modulo p à la fin seulement

Calcul du produit scalaire : deuxième méthode

Hypothèse : $p - 1 < 2^{M-1}$ et $N < 2^{M/2}$

Idée :

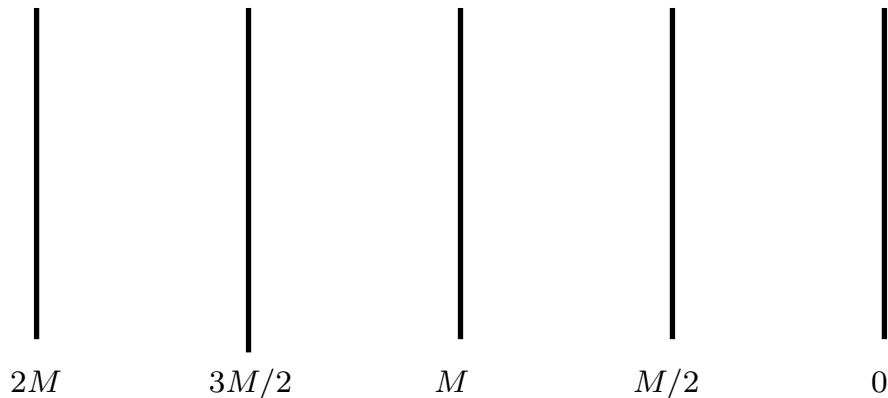
- Se ramener à des entiers représentables sur au plus une demi-mantisse
- Les sommer ensemble de manière exacte
- Réduire modulo p à la fin seulement

Utiliser `ExtractScalar` pour obtenir : $s_1 = \left\lfloor \frac{M}{2} \right\rfloor$

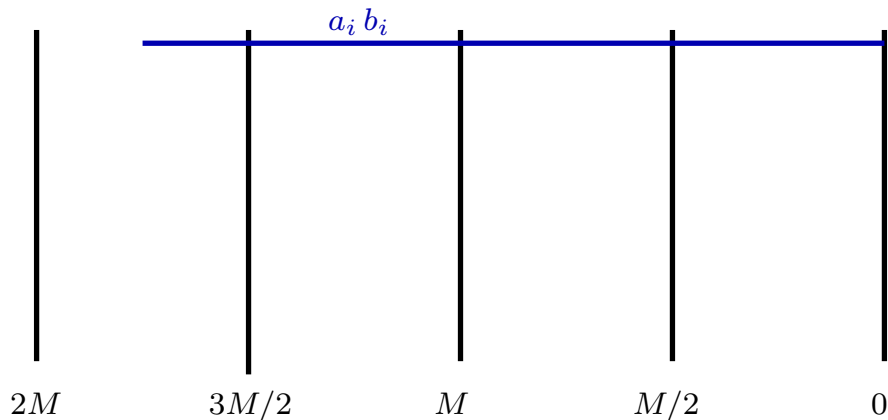
$$\forall i \in [1, N], \quad a_i b_i = \alpha_i + \beta_i + \gamma_i + \delta_i = A_i 2^{M+s_1} + B_i 2^M + C_i 2^{s_1} + D_i$$

$$a \cdot b = 2^{M+s_1} \sum_{i=1}^N A_i + 2^M \sum_{i=1}^N B_i + 2^{s_1} \sum_{i=1}^N C_i + \sum_{i=1}^N D_i \pmod{p}$$

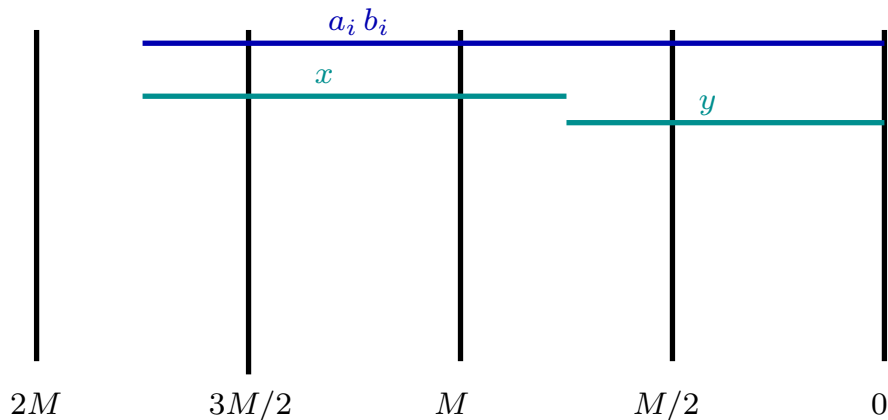
Deuxième méthode : principe de la découpe de $a_i b_i$ (1/2)



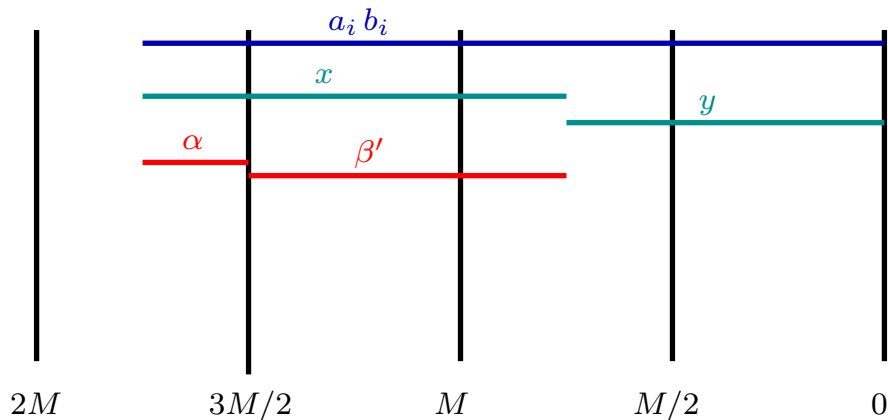
Deuxième méthode : principe de la découpe de $a_i b_i$ (1/2)



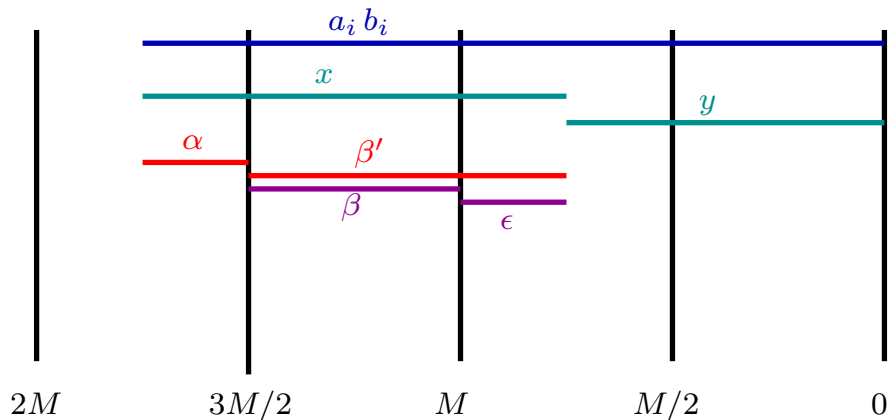
Deuxième méthode : principe de la découpe de $a_i b_i$ (1/2)



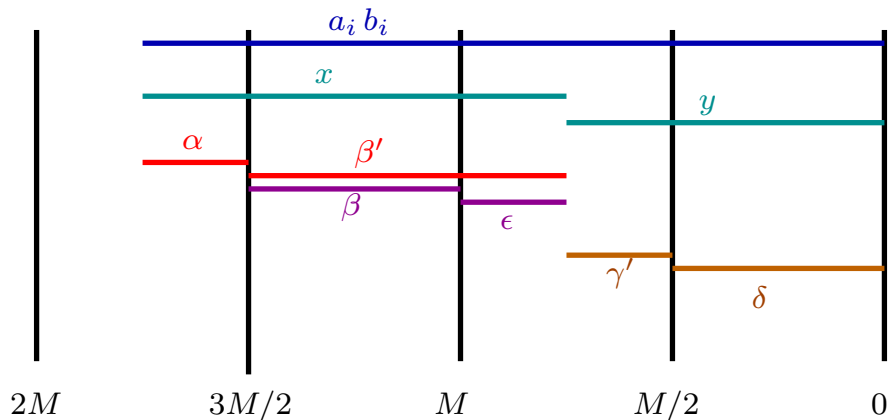
Deuxième méthode : principe de la découpe de $a_i b_i$ (1/2)



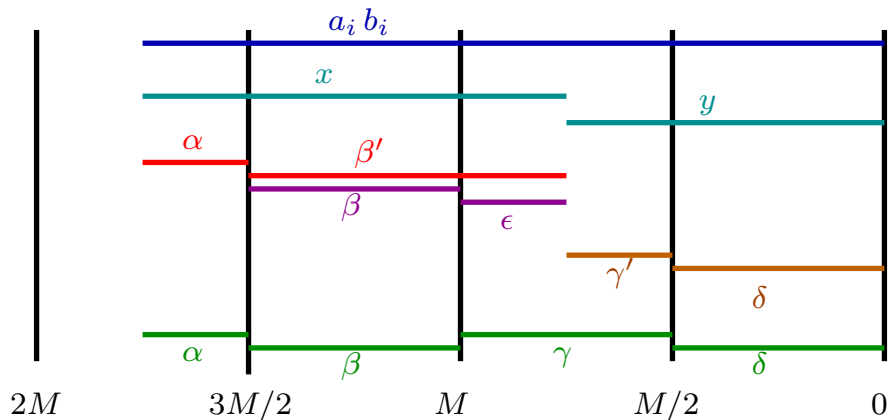
Deuxième méthode : principe de la découpe de $a_i b_i$ (1/2)



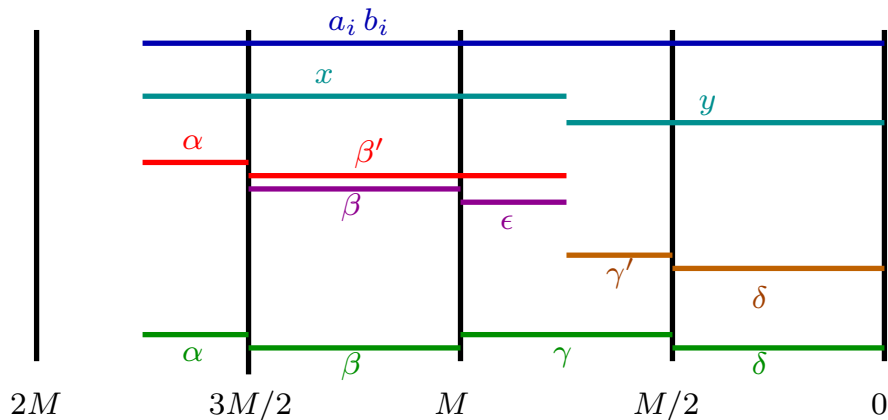
Deuxième méthode : principe de la découpe de $a_i b_i$ (1/2)



Deuxième méthode : principe de la découpe de $a_i b_i$ (1/2)



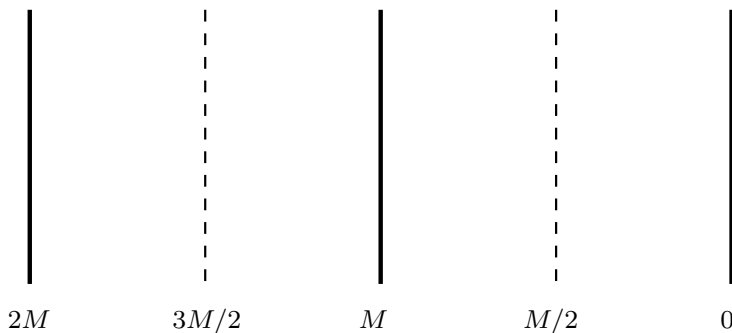
Deuxième méthode : principe de la découpe de $a_i b_i$ (1/2)



$$a_i b_i = \alpha + \beta + \gamma + \delta$$

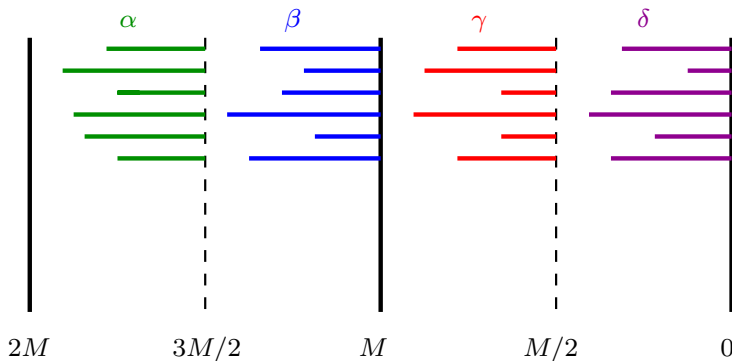
Deuxième méthode : principe de la découpe de $a_i b_i$ (2/2)

Découpe \longrightarrow 4 vecteurs de $N < 2^{M/2}$ éléments d'au plus $M/2$ bits



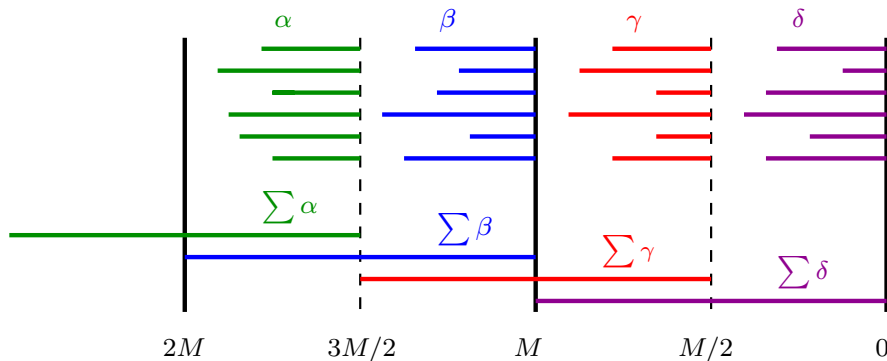
Deuxième méthode : principe de la découpe de $a_i b_i$ (2/2)

Découpe \longrightarrow 4 vecteurs de $N < 2^{M/2}$ éléments d'au plus $M/2$ bits



Deuxième méthode : principe de la découpe de $a_i b_i$ (2/2)

Découpe \longrightarrow 4 vecteurs de $N < 2^{M/2}$ éléments d'au plus $M/2$ bits



Deuxième méthode : Résultat

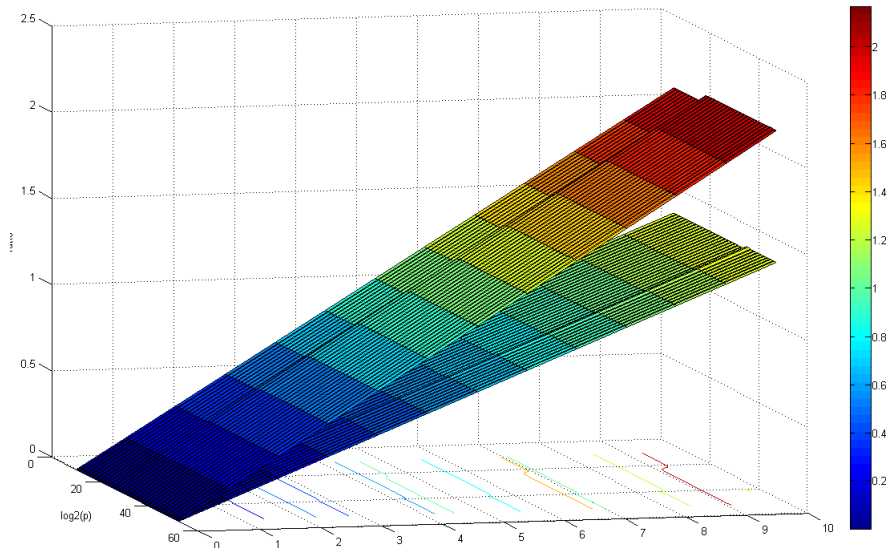
Résultat final :

$$a \cdot b = \sum_{i=1}^N \alpha_i + \sum_{i=1}^N \beta_i + \sum_{i=1}^N \gamma_i + \sum_{i=1}^N \delta_i \pmod{p}$$

Coût total : $16N + O(1)$ flops

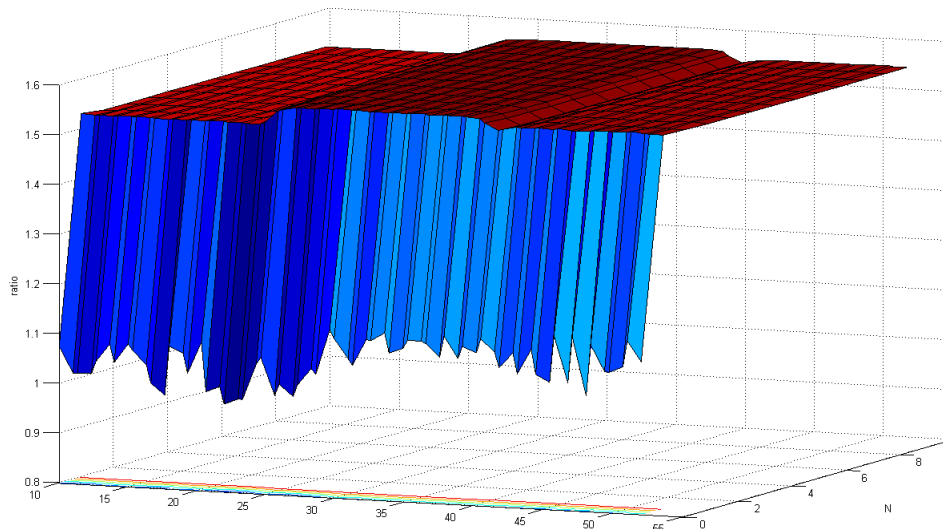
Deuxième méthode : Performances sur Itanium2 (1/3)

FIG.: Comparaison avec GMP : temps= $f(N, \log_2(p))$ — GMP en haut



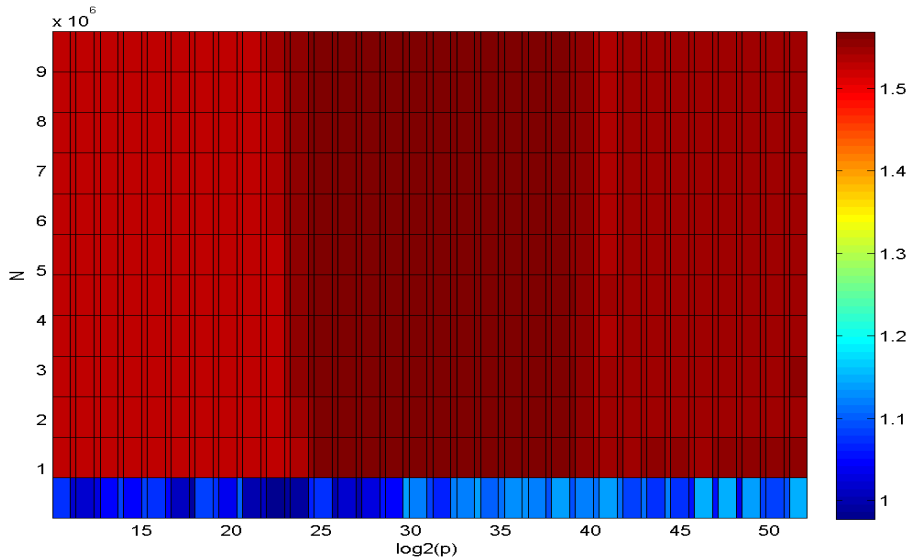
Deuxième méthode : Performances sur Itanium2 (2/3)

FIG.: Surface : $\text{ratio} = \text{temps}(GMP) / \text{temps}(\text{algo}) = f(N, \log_2(p))$



Deuxième méthode : Performances sur Itanium2 (3/3)

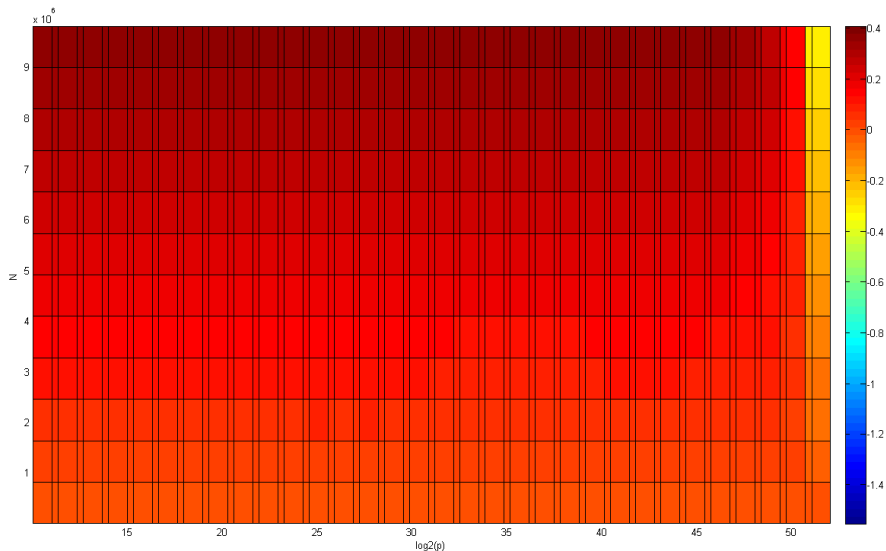
FIG.: $\text{ratio} = \text{temps}(GMP) / \text{temps}(algo) = f(N, \log_2(p))$



Comparaison des deux méthodes

Comparaison des deux méthodes

FIG.: $\text{ratio} = \text{temps}(\text{Methode}_2) - \text{temps}(\text{Methode}_1) = f(N, \log_2(p))$



Bilan

- Deux algorithmes efficaces pour le produit scalaire
- Méthodes efficaces par rapport à GMP
- Utilisation des transformations exactes en arrondi vers zéro

Perspectives :

- Deuxième méthode avec une découpe en 3 (avec $N < 2^{M/3}$)
- Étendre le corps premier $\mathbb{Z}/p\mathbb{Z}$ aux corps de Galois étendus $\text{GF}(2^n)$
- Paralléliser les algorithmes pour implantation sur GPU

Merci pour votre attention