Computation of dot products in finite fields with floating-point arithmetic

Stef Graillat

Joint work with Jérémy Jean

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

Computer-assisted proofs - tools, methods and applications
Dagstuhl Seminar 09471
Germany, November 15-20, 2009

Outline of the talk

- Motivations
- Basic
  - Floating-point arithmetic
  - Finite fields
- Computation of dot products
  - First method
  - Second method
- Comparison
- Conclusion and future work

Motivations

- Dot products: key tool in numerical linear algebra
- Fast algorithms in scientific computing
- Cryptology
- Error-correcting codes
- Computer algebra

Floating-point numbers

Normalized floating-point numbers $\mathbb{F} \subset \mathbb{R}$:

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

b : basis, M : precision, e : exponent such that $e_{\min} \le e \le e_{\max}$

Approximation of $\mathbb{R}$ by $\mathbb{F}$ with rounding $\mathbf{fl} : \mathbb{R} \to \mathbb{F}$.
Let $x \in \mathbb{R}$ then

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

Unit rounding $\mathbf{u} = b^{1-M}$ for rounding toward zero

Standard model of floating-point arithmetic

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

The result $x \circ y$ is not in general a floating-point number

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

IEEE 754 standard (1985)

Type	Size	Mantissa	Exponent	Unit rounding	Interval
Single	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	$\mathbf{u} = 2^{1-53} \approx 2,22 \times 10^{-16}$	$\approx 10^{\pm 308}$

Finite field $\mathbb{F}_p$ (p prime)

$\mathbb{F}_p = \mathbb{Z}/p\mathbb{Z} = GF(p) = \{0, 1, \ldots, p-1\}$ is a finite field with characteristic p

Finite field $\mathbb{F}_p$ (p prime)

$\mathbb{F}_p = \mathbb{Z}/p\mathbb{Z} = GF(p) = \{0, 1, \ldots, p-1\}$ is a finite field with characteristic p

Operations in the field, for $a, b \in \mathbb{Z}/p\mathbb{Z}$:

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

Finite field $\mathbb{F}_p$ (p prime)

$\mathbb{F}_p = \mathbb{Z}/p\mathbb{Z} = GF(p) = \{0, 1, \ldots, p-1\}$ is a finite field with characteristic p

Operations in the field, for $a, b \in \mathbb{Z}/p\mathbb{Z}$:

- Addition: $a + b \in \{0, \ldots, 2(p-1)\} \to a + b \pmod{p} \in \mathbb{Z}/p\mathbb{Z}$
- Multiplication: $ab \in \{0, \ldots, (p-1)^2\} \to ab \pmod{p} \in \mathbb{Z}/p\mathbb{Z}$

Reduction modulo p for $a \in \mathbb{Z}/p\mathbb{Z}$:

$$a \pmod{p} = a - \left\lfloor \frac{a}{p} \right\rfloor p = a - \lfloor a.invP \rfloor p$$

Aim

Let $p \geq 3$ a prime number and $(a_i)_i, (b_i)_i$ two vectors of N scalars in $\mathbb{Z}/p\mathbb{Z}$. We want to compute the dot product of a and b in $\mathbb{Z}/p\mathbb{Z}$:

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

Aim

Let $p \geq 3$ a prime number and $(a_i)_i, (b_i)_i$ two vectors of N scalars in $\mathbb{Z}/p\mathbb{Z}$. We want to compute the dot product of a and b in $\mathbb{Z}/p\mathbb{Z}$:

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

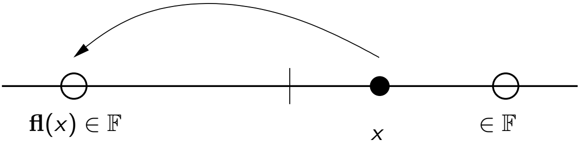
Assumptions:

- The integers are stored as floating-point numbers $\longrightarrow \mathbb{F} \cap \mathbb{N}$
- The prime p satisfies $\quad p - 1 < 2^{M-1}$
- The numbers are assumed to be nonnegative
- The rounding mode is rounding toward zero

Rounding toward zero in $\mathbb{R}^+$

Let $x \in \mathbb{R}^+$ $\quad$ $\mathbf{fl}(x)$ be the rounding toward zero of x in $\mathbb{F}$

- Equivalent to a truncation

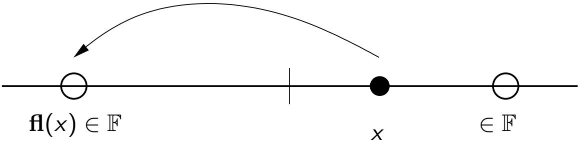


Rounding toward zero in $\mathbb{R}^+$

Let $x \in \mathbb{R}^+$ $\quad$ $\mathbf{fl}(x)$ be the rounding toward zero of x in $\mathbb{F}$

- Equivalent to a truncation
- The rounding is less or equal to the exact number:

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



Rounding toward zero in $\mathbb{R}^+$

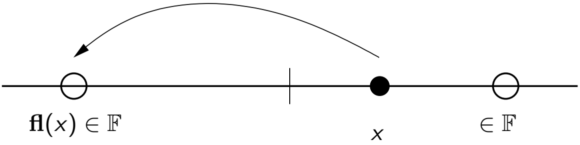
Let $x \in \mathbb{R}^+$ $\quad$ $\mathbf{fl}(x)$ be the rounding toward zero of x in $\mathbb{F}$

- Equivalent to a truncation
- The rounding is less or equal to the exact number:

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

- The rounding error is nonnegative:

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



Error-free Transformations (EFT)

Problem : the result of a floating-point operation is generally not representable by a floating-point numbers.

Solution: *Error-free transformations*

- non-evaluated sum of two floating-point numbers
  - the floating-point result of the operation
  - the rounding error (which is representable in $\mathbb{F}$ in our cases)
- For $a, b \in \mathbb{F} \cap \mathbb{N}$ and $\circ \in \{+\,,\times\}$,

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

  which is mathematically true.

Error-free Transformations for the product (1/2)

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

- FMA(a, b, c) is the rounding of $a \cdot b + c$

Algorithm 1 (EFT for the product of two floating-point numbers)

function $[x, y] = \texttt{TwoProductFMA}(a, b)$
  $x = \mathbf{fl}(a \cdot b)$
  $y = \texttt{FMA}(a, b, -x)$

The FMA is now included in the IEEE 754-2008 standard

Error-free Transformations for the product (2/2)

Theorem 1

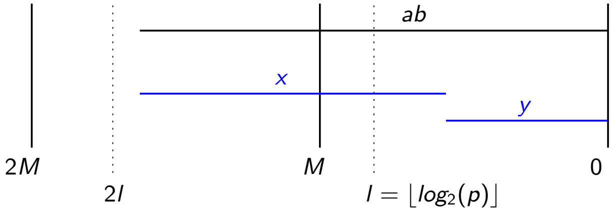
Let $a, b \in \mathbb{F} \cap \mathbb{N}$ *and* $x, y \in \mathbb{F}$ *such that*

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

Then

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

Algorithm `TwoProductFMA` *requires* 2 *flops.*



Binary euclidean division (1/2)

For $a, d \in \mathbb{F} \cap \mathbb{N}, d \neq 0$, the euclidean division of a by d is

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

For $a \in \mathbb{F} \cap \mathbb{N}$ and $\sigma = 2^k, \sigma \geq a$, one defines

Algorithm 2 (Split of a floating-point numbers)

function $[x, y] = \texttt{ExtractScalar}(\sigma, a)$
$q = \mathbf{fl}(\sigma + a)$
$x = \mathbf{fl}(q - \sigma)$
$y = \mathbf{fl}(x - a)$

fl is rounding toward zero

Algorithm first proposed by Rump, Ogita and Oishi in rounding to the nearest

Binary euclidean division (2/2)

Theorem 2

Let $a \in \mathbb{F} \cap \mathbb{N}$, $\sigma = 2^k, \sigma \geq a$ *and* $x, y \in \mathbb{F}$ *such that*

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

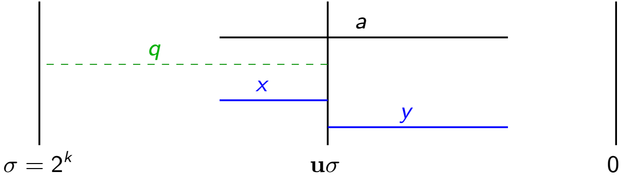
Then

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

Algorithm `ExtractScalar` *requires* 3 *flops.*

Remark:

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



Computation of dot products

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

Two different approaches

Computation of dot products

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

Two different approaches

- First method:

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

Computation of dot products

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

Two different approaches

- First method:

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

- Second method:

$$p - 1 < 2^{M-1} \qquad \text{but} \qquad N < 2^{M/2}$$

In `double`, the maximal vector size are $2^{53/2} \approx 10^8$.

First method

Computation of dot products: first method

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

Consequences :

- The sum of λ elements of the field can still be stored in the mantissa
- We can delay the reduction modulo p up to λ summations

Computation of dot products: first method

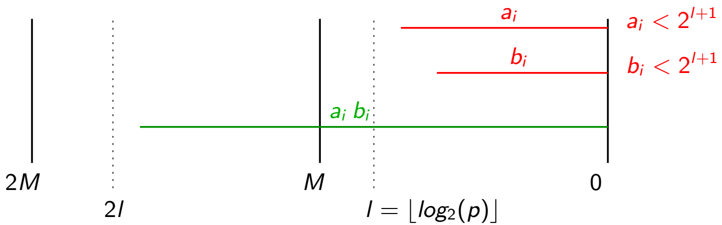
Assumption : $\lambda(p-1) < 2^{M-1}$

Consequences :

- The sum of λ elements of the field can still be stored in the mantissa
- We can delay the reduction modulo p up to λ summations

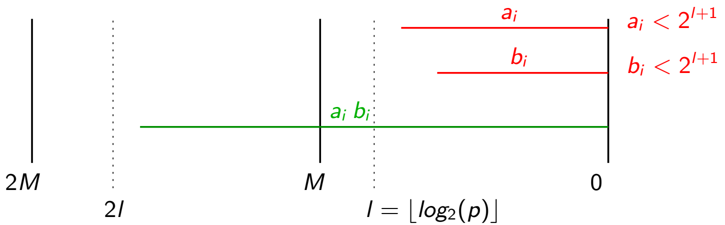
Jean-Guillaume Dumas: $\lambda(p-1)^2 < 2^M$

First method: principle



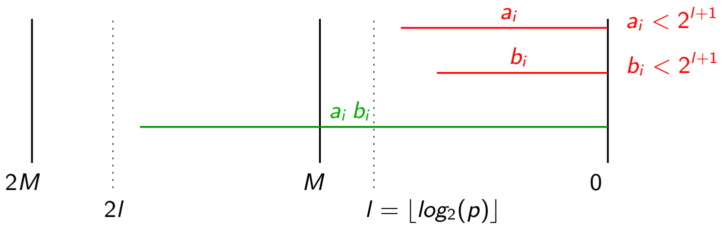
Let $l = \lfloor \log_2(p) \rfloor$

First method: principle



Let $l = \lfloor \log_2(p) \rfloor \quad \Longrightarrow \quad \mathbf{ufp}(p) := 2^l \neq p$ ($\mathbf{ufp}(p) =$ *most significant bit of p*)

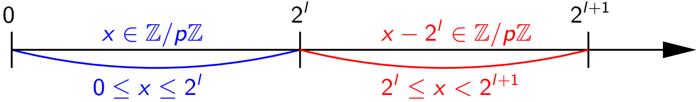
First method: principle



Let $l = \lfloor \log_2(p) \rfloor \quad \Longrightarrow \quad \mathbf{ufp}(p) := 2^l \neq p$ ($\mathbf{ufp}(p) =$ *most significant bit of p*)

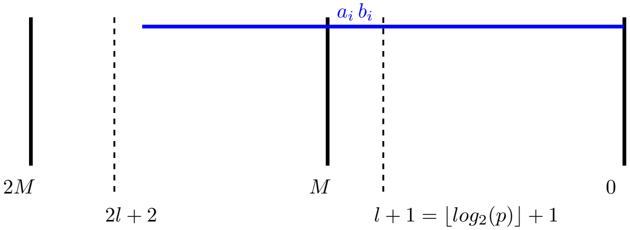
- $p \geq 3$ prime so: $\mathbf{ufp}(p) < p < 2.\mathbf{ufp}(p)$ i.e. $2^l < p < 2^{l+1}$
- Remarks:

$$\forall x \in [0, 2^{l+1} - 1] \cap \mathbb{F}, \quad \begin{cases} 0 \leq x \leq 2^l & \Longrightarrow x \in \mathbb{Z}/p\mathbb{Z} \\ 2^l < x < 2^{l+1} & \Longrightarrow x - 2^l \in \mathbb{Z}/p\mathbb{Z} \end{cases}$$



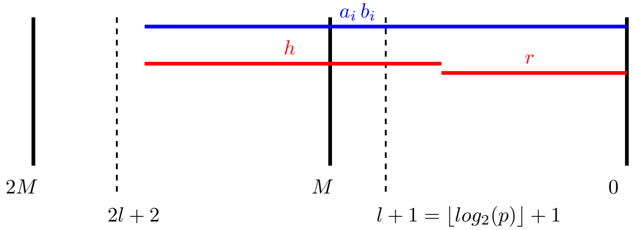
First method: principle

`TwoProductFMA` $\Longrightarrow$ $a_i\, b_i = h + r$



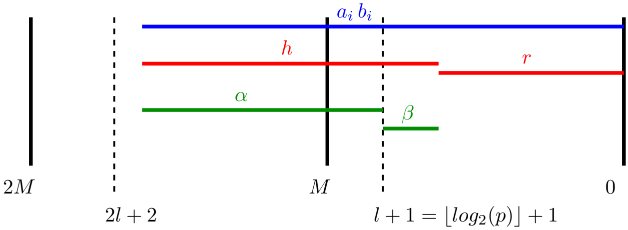
First method: principle

TwoProductFMA $\Longrightarrow$ $a_i\, b_i = h + r$



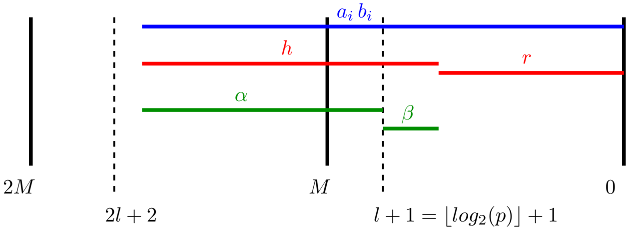
First method: principle

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



First method: principle

TwoProductFMA $\Longrightarrow$ $a_i\, b_i = h + r$

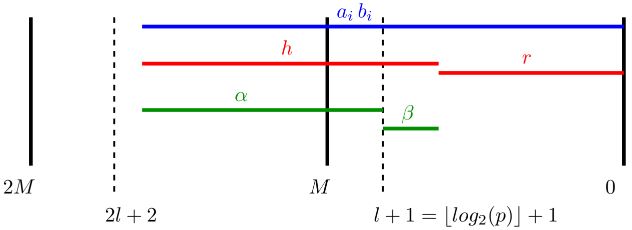


After splitting with ExtractScalar:

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

First method: principle

TwoProductFMA $\Longrightarrow$ $a_i\, b_i = h + r$

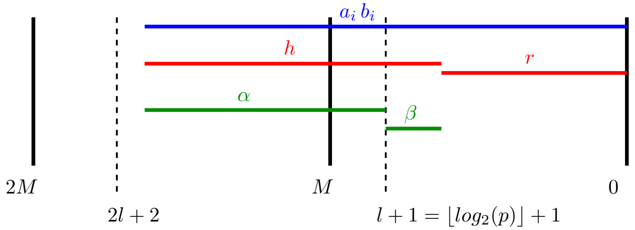


After splitting with ExtractScalar:

- $h = \alpha + \beta$ with $0 \leq \alpha/2^{l+1}, \beta < 2^{l+1}$
- We accumulate $\alpha/2^{l+1} \in \mathbb{Z}/p\mathbb{Z}$ or $\alpha/2^{l+1} - 2^l \in \mathbb{Z}/p\mathbb{Z}$
- We remember the number n_α of added -2^l

First method: principle

TwoProductFMA $\Longrightarrow$ $a_i\, b_i = h + r$



After splitting with ExtractScalar:

- $h = \alpha + \beta$ with $0 \leq \alpha/2^{l+1}, \beta < 2^{l+1}$
- We accumulate $\alpha/2^{l+1} \in \mathbb{Z}/p\mathbb{Z}$ or $\alpha/2^{l+1} - 2^l \in \mathbb{Z}/p\mathbb{Z}$
- We remember the number n_α of added -2^l
- Similar for β: $\beta \in \mathbb{Z}/p\mathbb{Z}$ or $\beta - 2^l \in \mathbb{Z}/p\mathbb{Z}$
- $n_\beta :=$ number of correction of -2^l for β

First method: final computation

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

First method: final computation

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

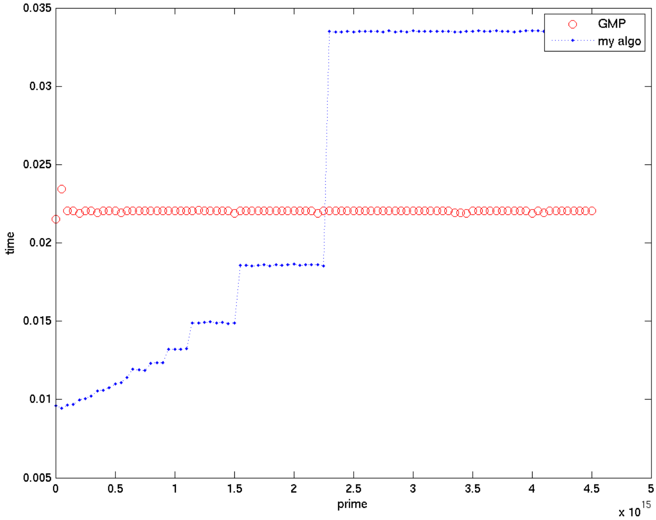
$\lambda(p-1) < 2^{M-1} \Longrightarrow$ summation by bundle of λ numbers and then reduction mod p

Performances

- On Itanium2
- With FMA
- In double precision ($p - 1 < 2^{53-1}$)
- Comparison with GMP

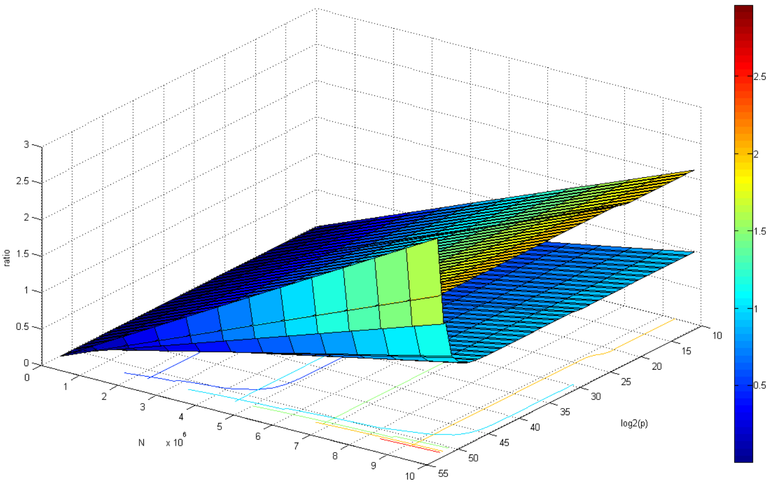
First method: Performances on Itanium2 (1/4)

Figure: Comparison with GMP: time=$f\left(p \in [2^{23}, 2^{52}]\right)$, for $N = 10^5$



First method: Performances on Itanium2 (2/4)

Figure: Comparison with GMP: time=$f(N, \log_2(p))$ —— GMP on the top



First method: Performances on Itanium2 (3/4)

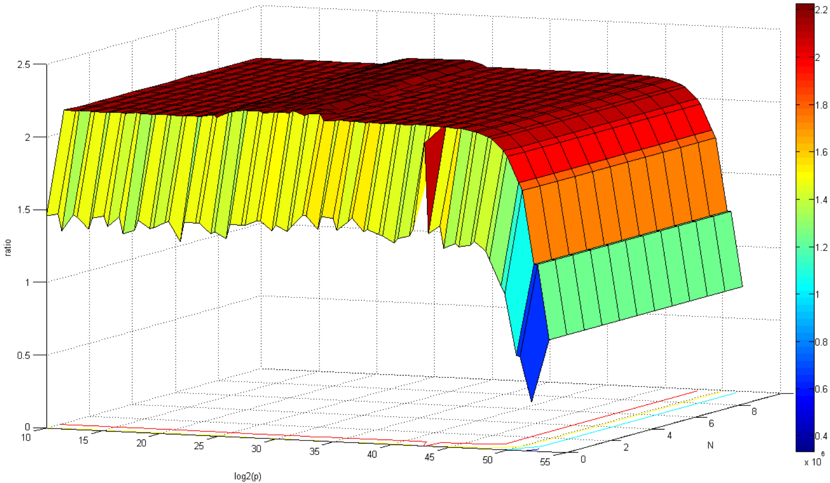
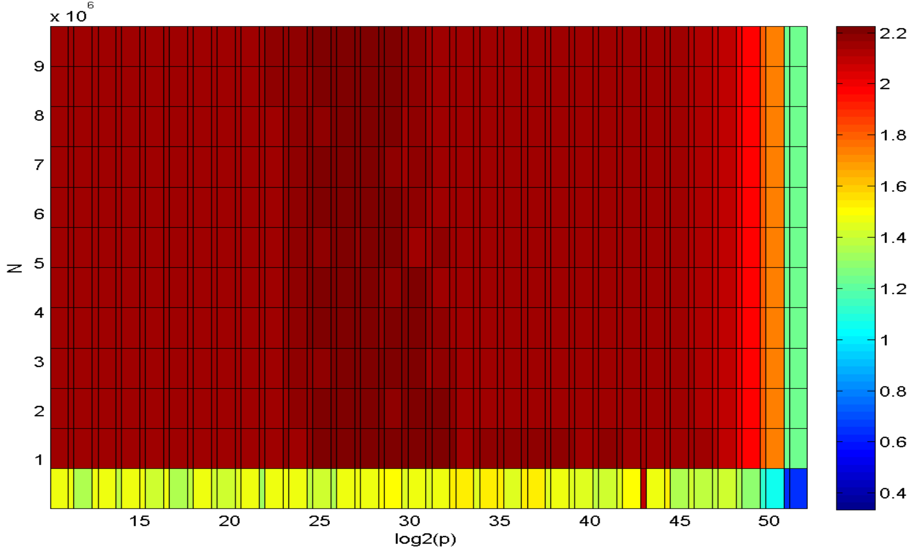


Figure: Surface: ratio=$time(GMP)/time(algo) = f(N, \log_2(p))$

First method: Performances on Itanium2 (4/4)

Figure: ratio= $time(GMP)/time(algo) = f(N, \log_2(p))$



Second method

Computation of dot products: second method

Assumption : $p - 1 < 2^{M-1}$ and $N < 2^{M/2}$

Computation of dot products: second method

Assumption : $p-1 < 2^{M-1}$ and $N < 2^{M/2}$

Idea :

- Split the number with a representation with only half the mantissa
- Sum them without error
- Reduction modulo p only at the end

Computation of dot products: second method

Assumption : $p - 1 < 2^{M-1}$ and $N < 2^{M/2}$

Idea :

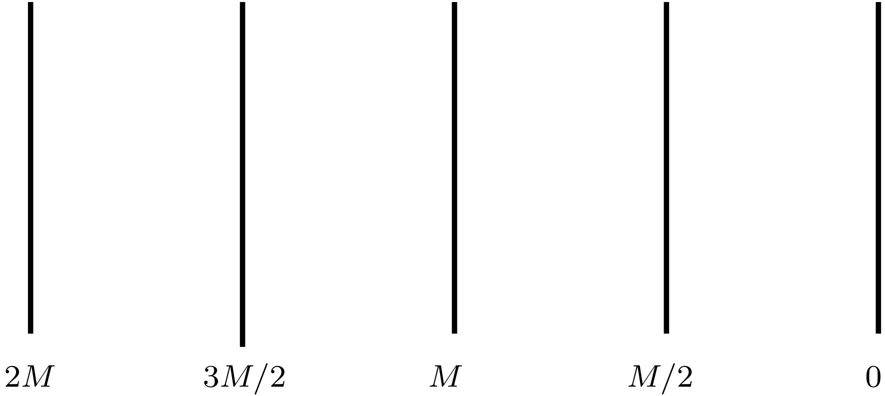
- Split the number with a representation with only half the mantissa
- Sum them without error
- Reduction modulo p only at the end

Use `ExtractScalar` to get: $s_1 = \left\lfloor \frac{M}{2} \right\rfloor$

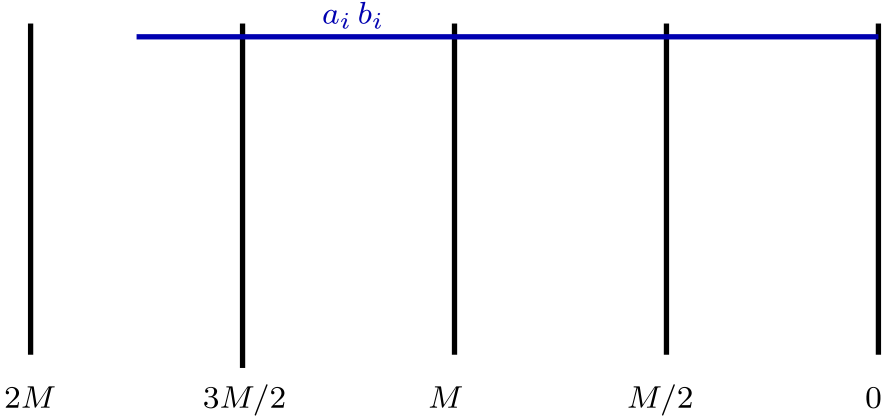
$$\forall i \in [1, N], \quad a_i\, b_i = \alpha_i + \beta_i + \gamma_i + \delta_i = A_i\, 2^{M+s_1} + B_i\, 2^M + C_i\, 2^{s_1} + D_i$$

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

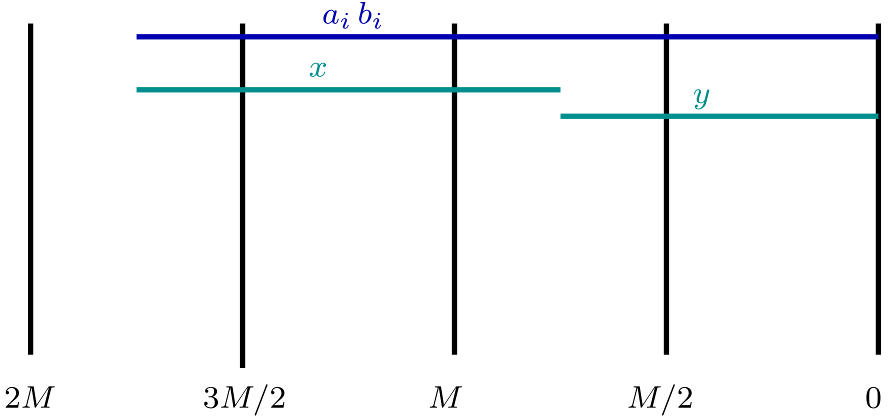
Second method: principle of the splitting of $a_i\, b_i$ (1/2)



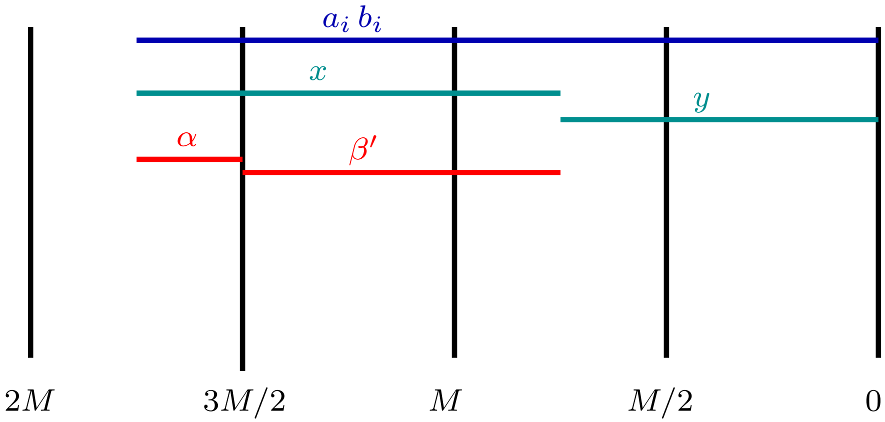
Second method: principle of the splitting of $a_i\, b_i$ (1/2)



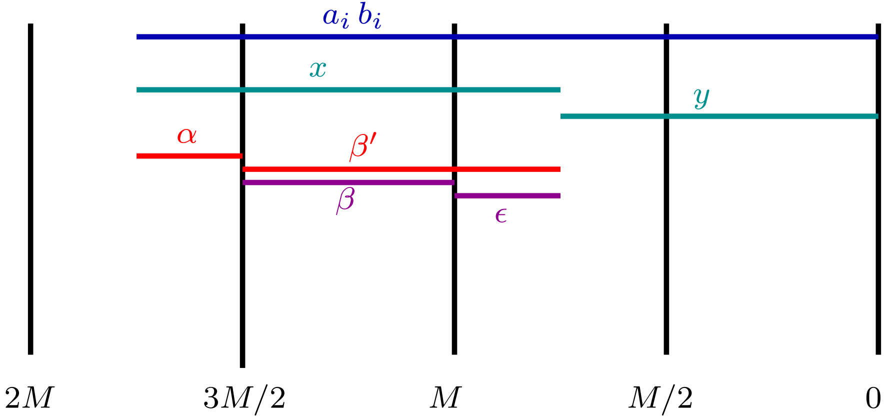
Second method: principle of the splitting of $a_i\, b_i$ (1/2)



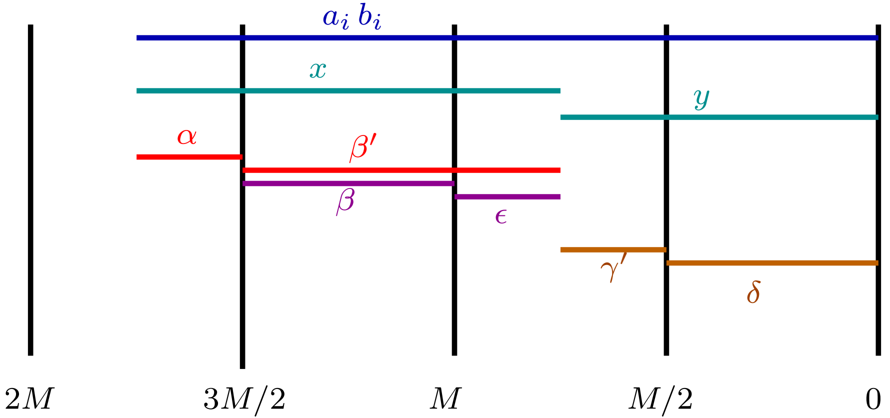
Second method: principle of the splitting of $a_i\, b_i$ (1/2)



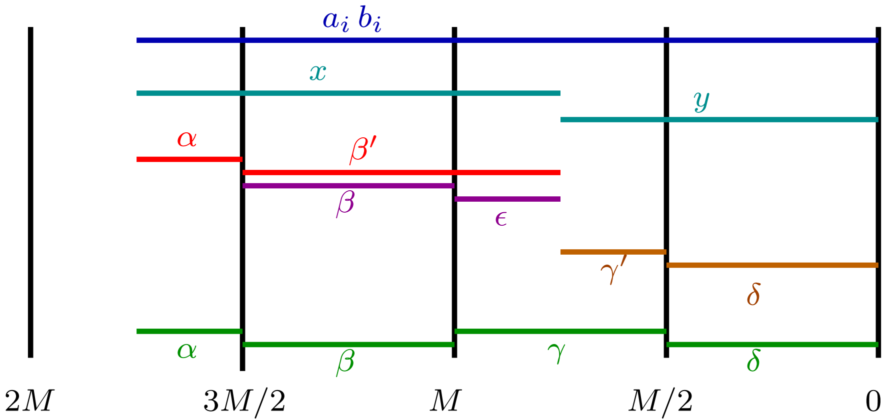
Second method: principle of the splitting of $a_i\, b_i$ (1/2)



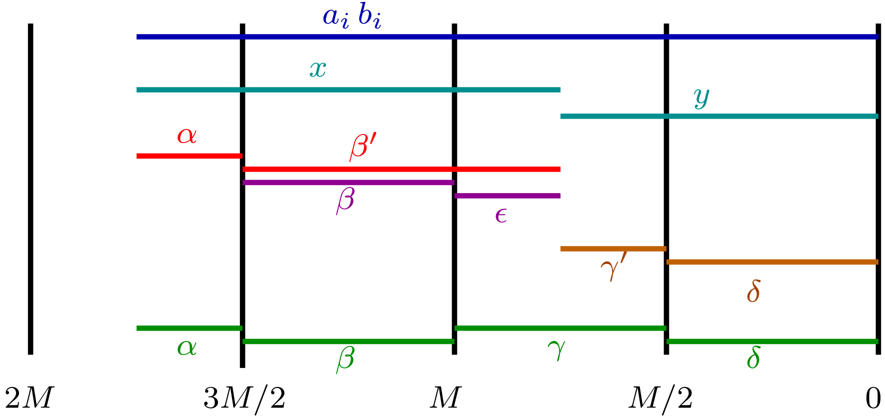
Second method: principle of the splitting of $a_i\, b_i$ (1/2)



Second method: principle of the splitting of $a_i\, b_i$ (1/2)



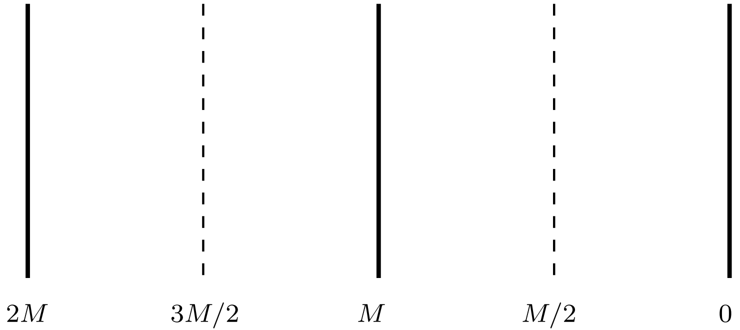
Second method: principle of the splitting of $a_i\, b_i$ (1/2)



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

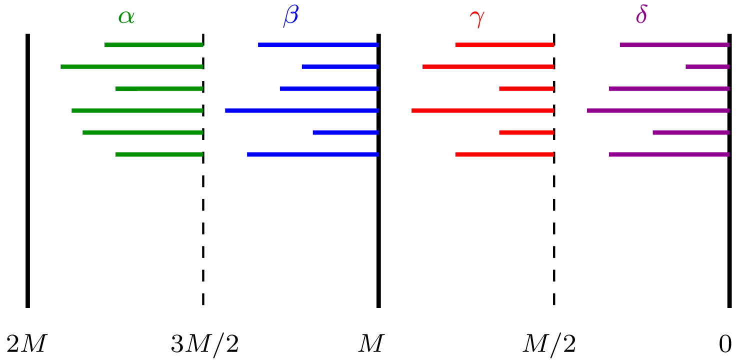
Second method: principle of the splitting of $a_i\, b_i$ (2/2)

Split $\longrightarrow$ 4 vectors of $N < 2^{M/2}$ elements with at most $M/2$ bits



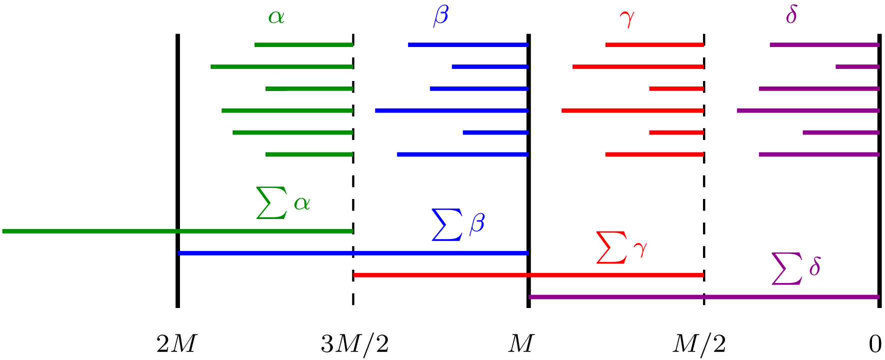
Second method: principle of the splitting of $a_i\, b_i$ (2/2)

Split $\longrightarrow$ 4 vectors of $N < 2^{M/2}$ elements with at most $M/2$ bits



Second method: principle of the splitting of $a_i\, b_i$ (2/2)

Split $\longrightarrow$ 4 vectors of $N < 2^{M/2}$ elements with at most $M/2$ bits



Second method: Results

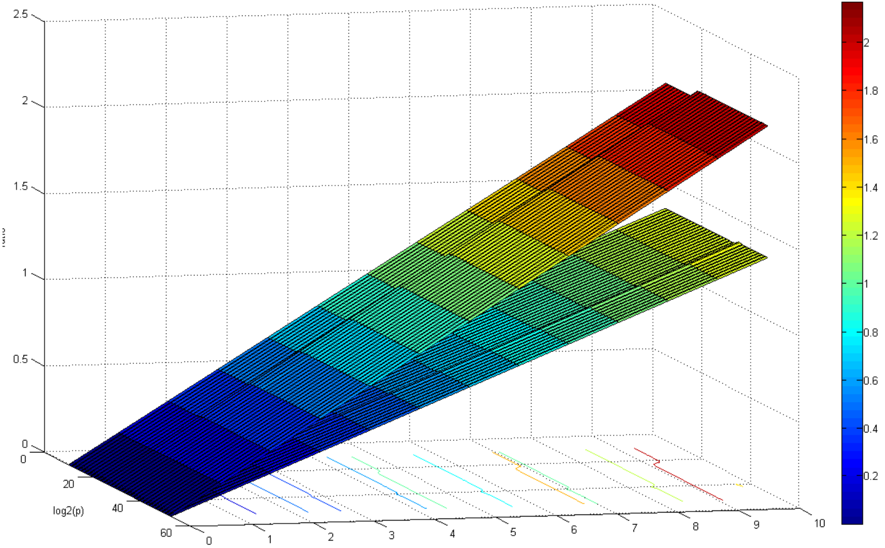
Final results:

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

Total cost: $16N + O(1)$ flops

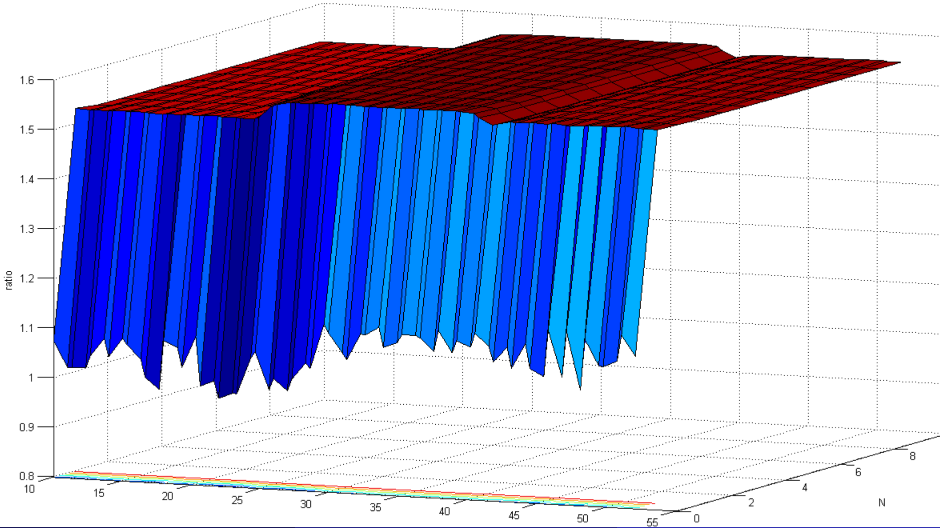
Second method: Performances on Itanium2 (1/3)

Figure: Comparison with GMP: time$=f(N, \log_2(p))$ —— GMP on the top



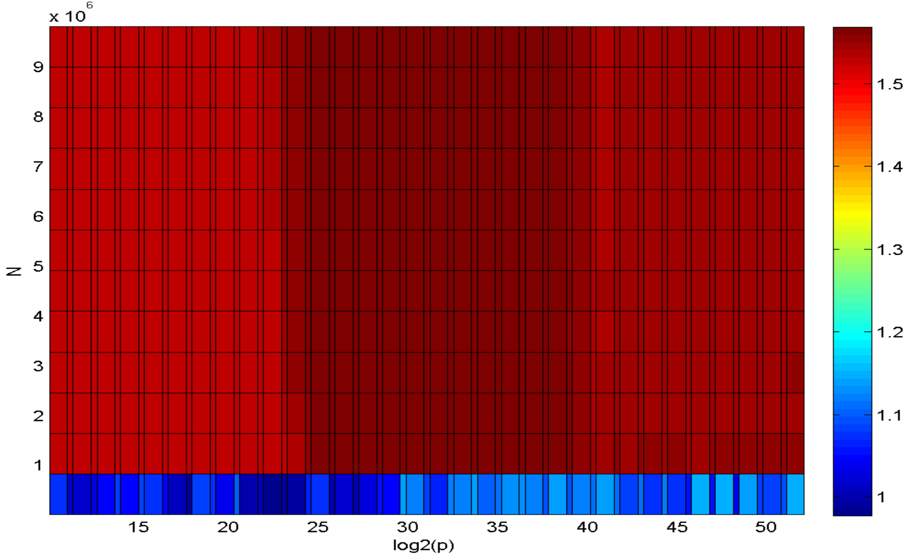
Second method: Performances on Itanium2 (2/3)

Figure: Surface: ratio=$time(GMP)/time(algo) = f(N, \log_2(p))$



Second method: Performances on Itanium2 (3/3)

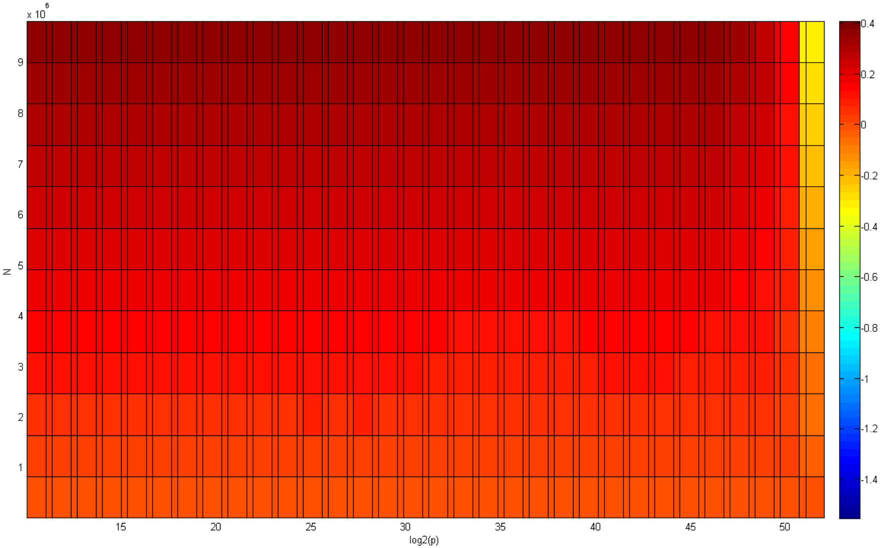
Figure: ratio$=time(GMP)/time(algo) = f(N, \log_2(p))$



Comparison of the two methods

Comparison of the two methods

Figure: $time(Method_2) - time(Method_1) = f(N, \log_2(p))$



Conclusion and future work

Conclusion:

- Two efficient algorithms for computing dot product
- Efficient algorithms compared to GMP
- Use of error-free transformations in rounding toward zero

Future work:

- Second method with a splitting in 3 parts (with $N < 2^{M/3}$)
- Extension to Galois fields $GF(2^n)$
- Use of `longlong` library
- Use of RNS techniques
- Parallelisation of the algorithms for GPU

The end

Thank you for your attention