Étude et conception d'opérateurs arithmétiques

Habilitation à diriger les recherches
Université Rennes 1

Arnaud Tisserand

CNRS, IRISA, équipe-projet CAIRN

6 juillet 2010

Plan

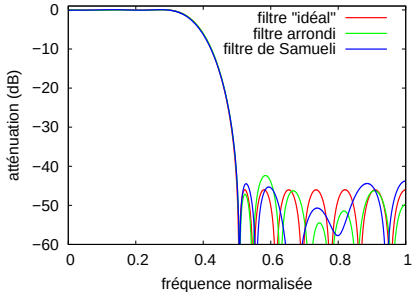
- Introduction
- Résumé des activités
- Sélection de travaux :
  - Architecture reconfigurable pour l'arithmétique en ligne
  - Multiplication par des constantes
  - Opérateurs arithmétiques sécurisés et bibliothèque arithmétique pour la cryptographie
- Conclusion et perspectives

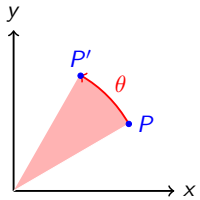
Exemples d'opérateurs arithmétiques (1/3)

```c
#include <stdio.h>
#include <math.h>

typedef float func_t(float x);

float integrate(func_t *f, float a, float b, int n)
{
  int i;  float x = a;  float s = 0.0;  float h = (b-a)/(n-1);
  for (i=0 ; i<n ; i++) { s += f(x)*h;  x += h; }
  return s;
}

float f(float x) { return 1.0/x; }

int main (void)
{
  float r, rth=0.69314718055994530942;

  r = integrate((func_t*)f, 1.0, 2.0, 1000);

  printf("r=%f      rth=%f      err=%f\n", r, rth, fabs(rth-r));
  return 0;
}
```


Exemples d'opérateurs arithmétiques (2/3)

$$y[n] = \sum_{i=0}^{N} c_i \times x[n-i]$$

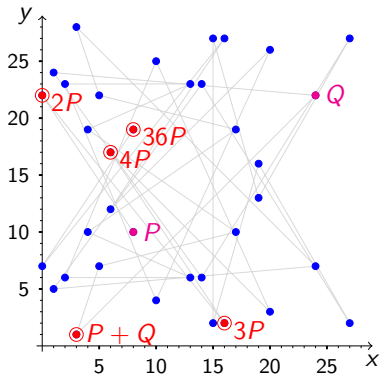




$$\begin{pmatrix} x' \\ y' \end{pmatrix} = \begin{pmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix}$$

Exemples d'opérateurs arithmétiques (3/3)

Opérations sur la courbe elliptique $y^2 = x^3 + 4x + 20$ sur le corps fini $\mathbb{F}_{29}$



addition de 2 points : $P + Q$
doublement d'un point : $[2]P$

résultat : le point (x_3, y_3) avec
$x_3 = \lambda^2 - x_1 - x_2, \quad y_3 = \lambda(x_1 - x_3) - y_1$

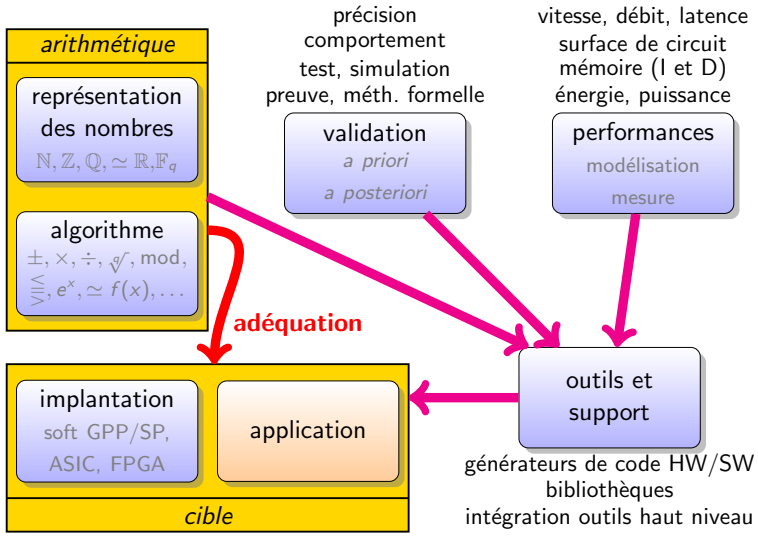
$$\lambda = \begin{cases} \dfrac{y_2 - y_1}{x_2 - x_1} & \text{si } P \neq \pm Q \\ \dfrac{3x_1^2 + a}{2y_1} & \text{si } P = Q \end{cases}$$

opérations sur le corps fini : $\pm, \times, x^{-1}$

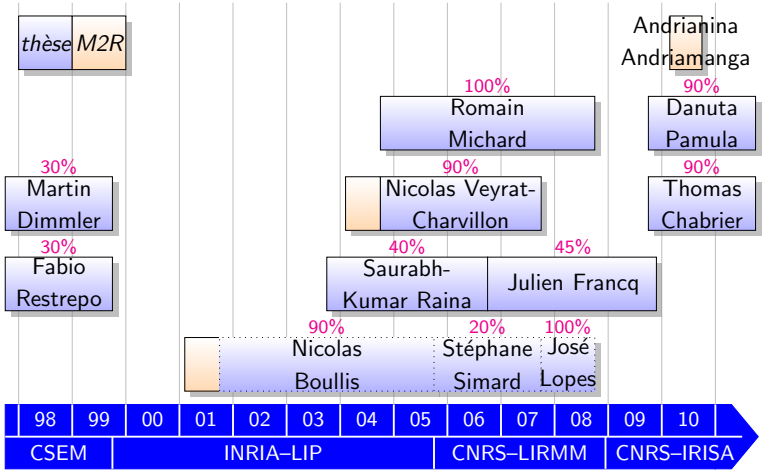
$P = (8, 10)$, $Q = (24, 22)$

multiplication scalaire ($k \in \mathbb{N}$, grand) :
$[k]P = \underbrace{P + P + \ldots + P}_{k \text{ fois}}$

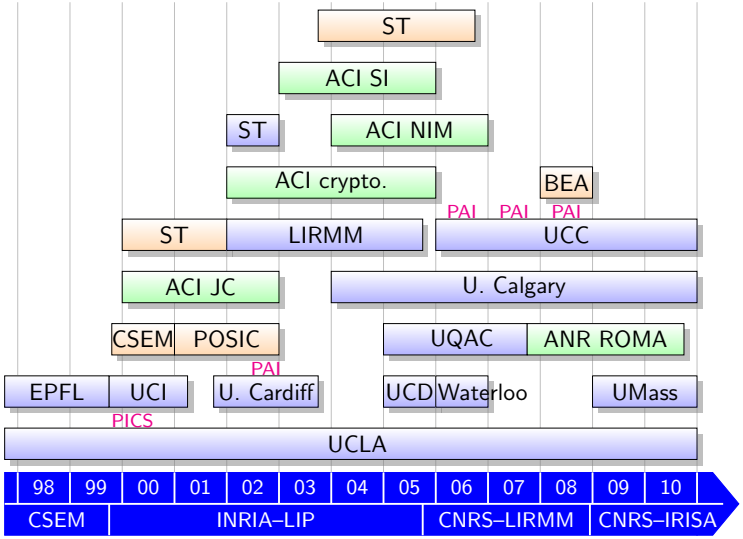
Arithmétique des ordinateurs, une « définition »



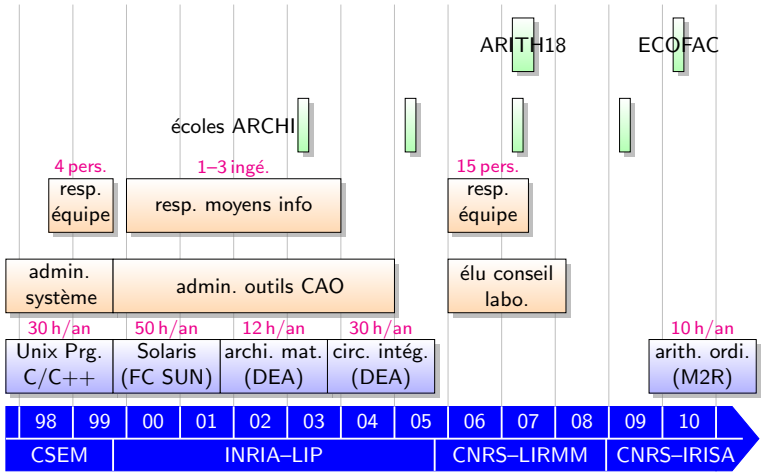
Résumé des activités : encadrement



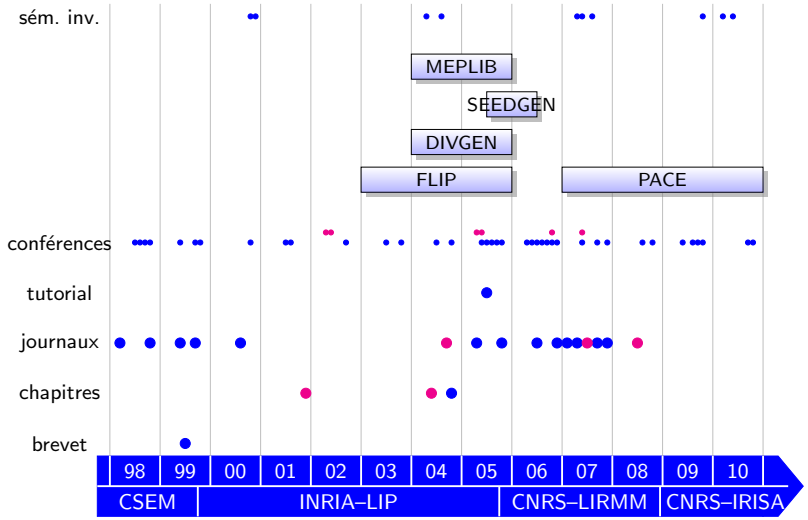
Résumé des activités : collaborations



Résumé des activités : enseignement & animation



Résumé des activités : production scientifique



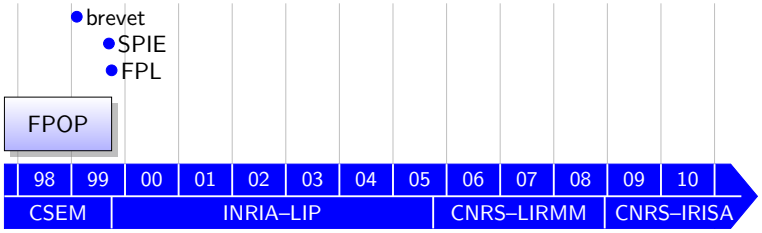
Thèmes de recherche abordés

- Arithmétique en ligne
- Architectures reconfigurables pour le calcul : FPOP et FPPA
- Opérateurs spécifiques pour FPGA
- Multiplication : par des constantes, tronquée, x^n ($n \in \mathbb{N}$)
- Division : par une constante, asynchrone, générateur automatique
- Approximation de fonctions : tables ou polynômes
- Bibliothèques flottantes : CoolRISC et FLIP (ST200)
- Arithmétique à basse consommation d'énergie : opérateurs, modélisation, évaluation, GPU
- Opérateurs arithmétiques pour la cryptographie : corps finis, hachage, sécurisation contre les attaques, bibliothèque PACE
- Outils : générateurs, programmation, gestion de la précision
- Arithmétique par estimation
- Générateurs vraiment aléatoires

Architecture reconfigurable pour l'arithmétique en ligne

Généralités :

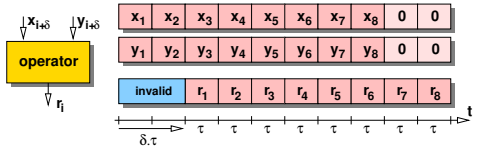
- FPOP : *Field Programmable On-line oPerators*
- Travail avec P. Marchal, P. Nussbaum et C. Piguet
- Cible : applications embarquées de contrôle numérique



Arithmétique en ligne (1/2)

Arithmétique en ligne :

- arithmétique série
- poids forts en tête



Avantages :

- faible surface
- parallélisme (pipeline)
- haut débit
- circuit régulier
- interconnexion simple
- précision variable possible assez simplement

Inconvénients :

- opérations lentes
- synchronisation lourde
- peu (pas) d'outil

Origine : M. Ercegovac dans les années 70

Arithmétique en ligne (2/2)

Représentation redondante des nombres :

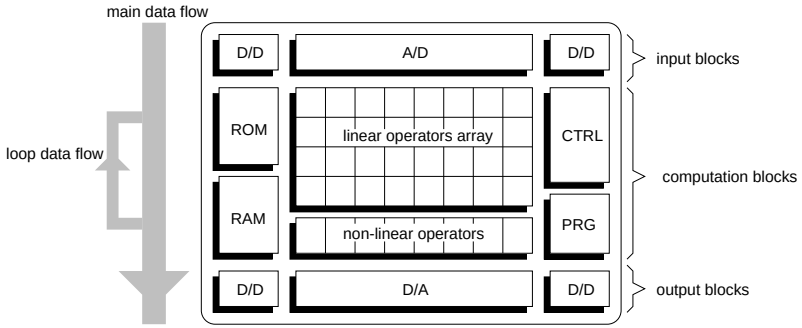
$$X = \sum_{i=-m}^{n} x_i \beta^i \qquad x_i \in \mathcal{D}$$

- toutes les opérations avec les poids forts en tête
- certaines opérations à délai en ligne δ (latence) variable (p. ex. $\lessgtr$)
- bases utilisées :
  - base $\beta = 2$: chiffres dans $\mathcal{D}_{\text{bs}} = \{-1, 0, 1\}$ (*borrow-save*)
  - base $\beta = 4$: chiffres dans $\mathcal{D}_{4,3}\{-3, \ldots, 3\}$ ou $\mathcal{D}_{4,2}\{-2, \ldots, 2\}$

Caractéristiques des principales opérations :

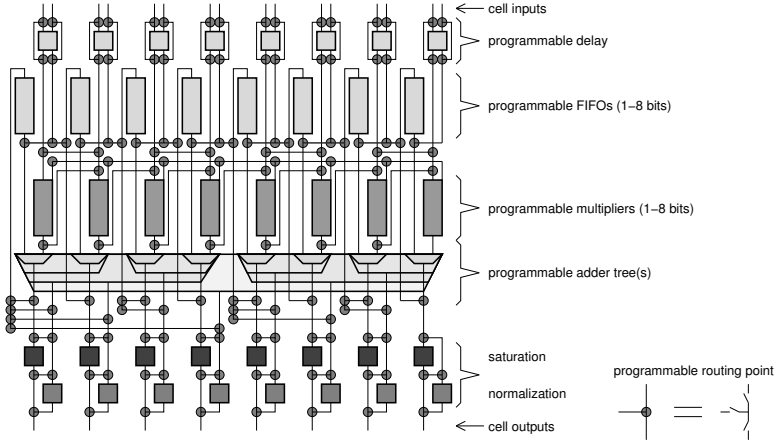
opération	$\pm$	$\times$	$\div$	$\sqrt{}$
surface	$\mathcal{O}(1)$	$\mathcal{O}(n)$	$\mathcal{O}(n)$	$\mathcal{O}(n)$
temps calcul	$\mathcal{O}(n)$	$\mathcal{O}(n)$	$\mathcal{O}(n)$	$\mathcal{O}(n)$

FPOP : architecture générale

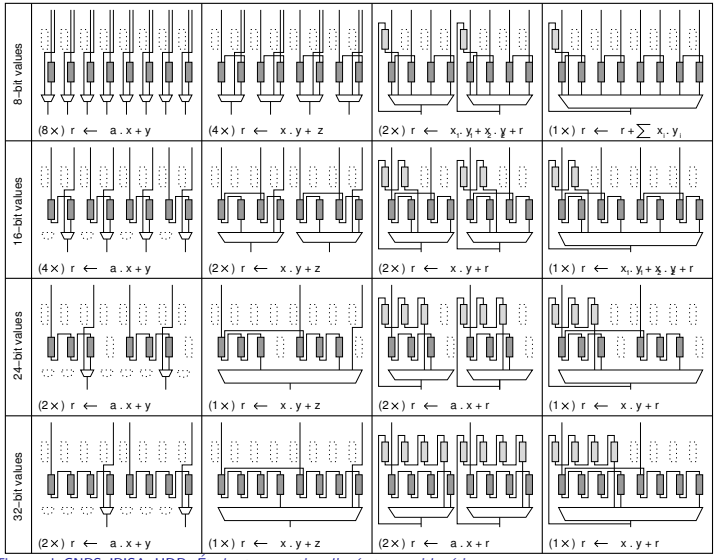


- blocs d'opérations linéaires $c \cdot x$ et configuration pour :
  - $c \cdot x + y$, $\sum_{i=0}^{k} c_k \cdot x_k$
  - multiplication $x \cdot y$, $x \cdot y + z$
- autres opérations : $x \div y, 1/x, \sqrt{x}$ et approx. fonction $f(x) \simeq p(x)$

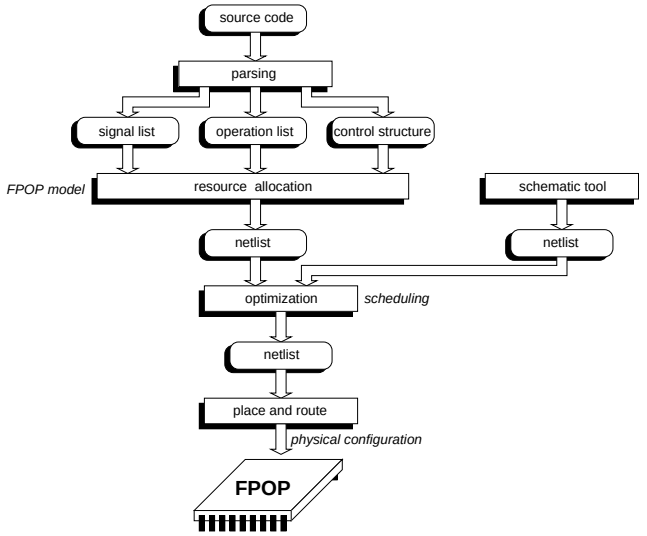
FPOP : cellule linéaire



FPOP : configurations d'une cellule linéaire



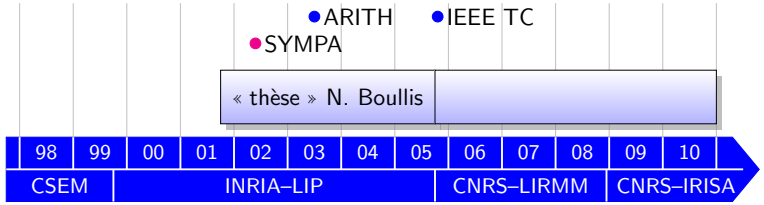
FPOP : flot de programmation



Multiplication par des constantes

Généralités :

- Travail commencé avec la thèse de N. Boullis
- Générateur automatique de code VHDL très optimisé
- Résultats utilisés dans :
  - d'autres travaux et publications (p. ex. approx. de fonctions, TNS)
  - des collaborations : UQAC, U. Calgary
  - des aides ponctuelles



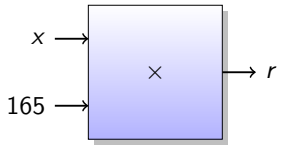
Multiplication par des constantes : problème

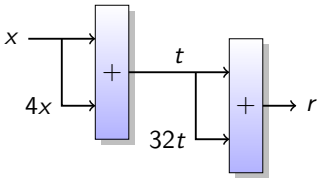
Idée générale : remplacer les multiplications par des constantes par une suite d'additions (ou soustractions) et de décalages

Buts :

- augmenter la vitesse
- réduire la taille et la consommation

Exemple simple : $r = 165 \cdot x$ $\quad 165 = (10100101)_2$





$t = x + 4 \cdot x$ $\quad r = t + 32 \cdot t$

$t = x + x \ll 2$ $\quad r = r + r \ll 5$

Multiplication par des constantes : variantes

Multiplication par une seule constante :

$$r = c \cdot x$$

Multiplication par un vecteur constant :

$$\begin{aligned} r_1 &= c_1 \cdot x \\ r_2 &= c_2 \cdot x \\ &\vdots \\ r_n &= c_n \cdot x \end{aligned}$$

Multiplication par une matrice constante :

$$\begin{aligned} r_1 &= c_{1,1} \cdot x_1 + c_{1,2} \cdot x_2 + \cdots + c_{1,m} \cdot x_m \\ r_2 &= c_{2,1} \cdot x_1 + c_{2,2} \cdot x_2 + \cdots + c_{2,m} \cdot x_m \\ &\vdots \\ r_n &= c_{n,1} \cdot x_1 + c_{n,2} \cdot x_2 + \cdots + c_{n,m} \cdot x_m \end{aligned}$$

Aspects temporels :

$$y[n] = \sum_{i=0}^{N} c_i \cdot x[n-i]$$

Multiplication par des constantes : algorithmes

Méthodes : recodages, recherche avec des fonctions de coût (p. ex. Bernstein), recherche de motifs, algorithmes génétiques...

Exemple : $p = 111463 \cdot x$

algo.	$p = 111463 \cdot x =$	#op.
direct	$(x \ll 16)+(x \ll 15)+(x \ll 13)+(x \ll 12)+(x \ll 9)$ $+(x \ll 8)+(x \ll 6)+(x \ll 5)+(x \ll 2)+(x \ll 1)+x$	10 ±
CSD	$(x \ll 17)-(x \ll 14)-(x \ll 12)+(x \ll 10)$ $-(x \ll 7)-(x \ll 5)+(x \ll 3)-x$	7 ±
Bernstein	$(((t_2 \ll 2)+x) \ll 3)-x$ où $t_1 = (((x \ll 3)-x) \ll 2)-x$ $t_2 = t_1 \ll 7+t_1$	5 ±
notre algo.	$(t_2 \ll 12)+(t_2 \ll 5)+t_1$ où $t_1 = (x \ll 3)-x$ $t_2 = (t_1 \ll 2)-x$	4 ±

CSD *canonical signed digit*, $111463 = 11011001101100111_2 = 100\bar{1}0\bar{1}0100\bar{1}0\bar{1}0100\bar{1}_2$

Multiplication par des constantes : résultats

FIR $(1, 5, 5, 1)$

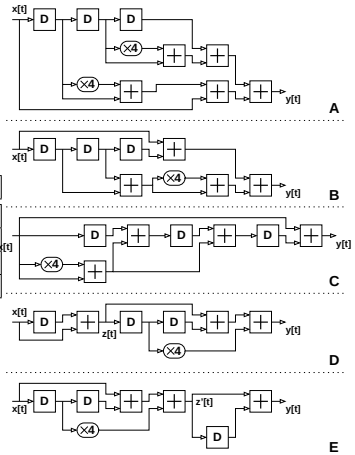
opérateur	init.	[1]	[2]	notre
DCT 8b	300	94	73	56
DCT 12b	368	100	84	70
DCT 16b	521	129	114	89
DCT 24b	789	212	—	119

Codes correcteurs :

opérateur	init.	[1]	[2]	notre
8×8 Had.	56	24	—	24
$(16, 11)$ R.-M.	61	43	31	31
$(15, 7)$ BCH	72	48	47	44
$(24, 12, 8)$ Golay	76	—	47	45

Filtres :

opérateur	init.	[22]	notre
8 bits	35	32	24
16 bits	72	70	46



Multiplication par des constantes : bilan

Résultats :

- optimisation de multiplications par des matrices constantes
- approche calcul exact
- générateur automatique de code VHDL très optimisé
- gains importants dans certains cas :
  - $\times$ 2 à 3 en vitesse
  - $\div$ 2 à 5 en surface
  - $\div$ 2 à 3 en consommation
- utilisé dans plusieurs études

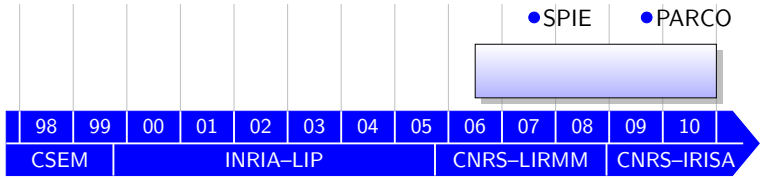
Travaux en cours et à venir :

- gestion de la précision (calcul approché)
- extension aux corps finis
- ajout contrainte compromis vitesse–surface

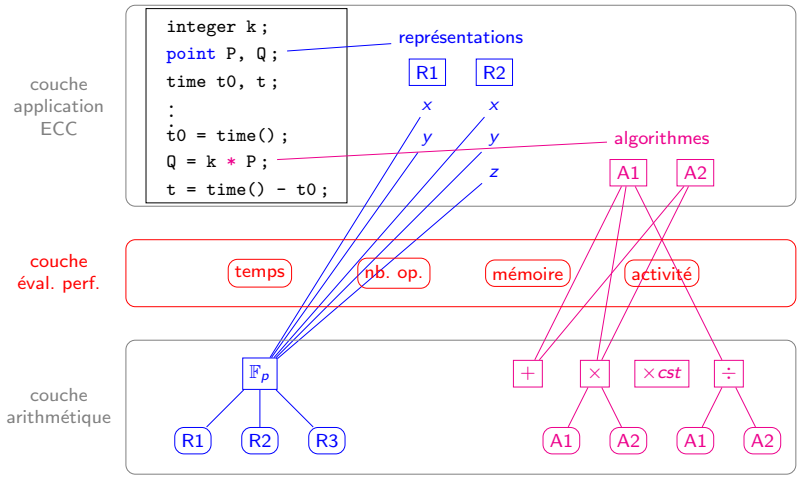
Bibliothèque PACE

Généralités :

- PACE = *Prototyping Arithmetic for Crypto Easily*
- Collaboration ARITH–LIRMM : P. Giorgi, L. Imbert et T. Izard
- Bibliothèque C++, licence LGPL, *template metaprogramming* : généricité + spécialisation pour les performances (*traits*)
- Motivations :
  - support mathématique limité dans les langages et processeurs
  - comparison de plusieurs solutions très difficile
    - temps pour coder toutes les solutions à tester
    - besoin d'un support de test uniforme et précis
  - validation rapide de nouvelles idées (algorithmes et représentations)



Bibliothèque PACE : architecture



Bibliothèque PACE : couche arithmétique (1/2)

```
integer<100> p = 29;
typedef gfp<100, p> fp_29;
fp_29 x = 17, y = 20, z;
z = x + y;    assert(z == 8);
cout << x << " + " << y << " = " << z << endl;
z = x - y;    assert(z == 26);
cout << x << " - " << y << " = " << z << endl;
z = x * y;    assert(z == 21);
cout << x << " * " << y << " = " << z << endl;
z = inv(x);   assert(z == 12);
cout << x << " ^(-1) = " << z << endl;
```

produit

```
17 + 20 = 8
17 - 20 = 26
17 * 20 = 21
17 ^(-1) = 12
```


Bibliothèque PACE : couche arithmétique (2/2)

Buts :

- support de plusieurs représentations et algorithmes
- petites modifications dans le code
- comparaison de plusieurs solutions

Exemple :

```
integer<600>                    a,  b,  c;
integer<600, RP_integer_GMP> a2, b2, c2;
...
c  = a  + b;
c2 = a2 + b2;
assert(c == c2);
```

État actuel :

- grands entiers : représentations multiples, opérations : $\pm, \times, {}^2, \div, \sqrt{}, {}^{-1}$, *mod*, *invmod*, *powmod*, *cmp*, *bit*, *popcount*...
- éléments de $\mathbb{F}_p$ (p quelconque) : représentations multiples, opérations : $\pm, \times, {}^2, {}^{-1}, =$, *bit*, *popcount*...

Futur : $\mathbb{F}_p$ avec des p particuliers, $\mathbb{F}_2$ et extensions, polynômes

Bibliothèque PACE : couche application ECC

```
integer<100> p = 29; typedef gfp<100, p> fp_29;
curve<fp_29> E(4, 20);
typedef point_aff<fp_29, E> point;
E.info();
point P1(5,22); point P2(16,27);
cout << "P1 = " << P1 << " P2 = " << P2 << endl;
point P3 = P1 + P2;
cout << "P1 + P2 = " << P3 << endl;
point P4 = 2 * P1;
cout << "[2] P1 = " << P4 << endl;
```

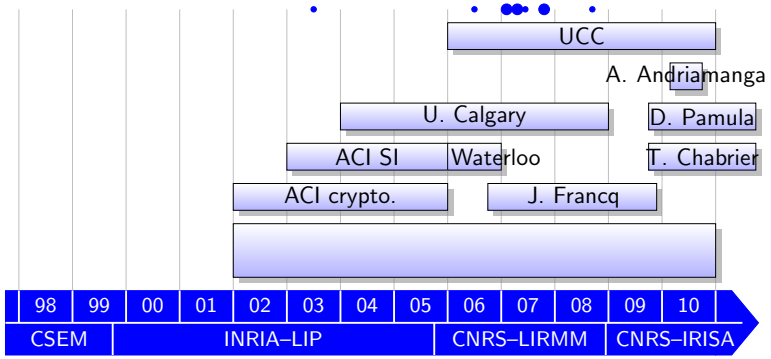
produit

```
Elliptic curve defined by y^2 = x^3 + 4*x + 20
P1 = (5 , 22) P2 = (16 , 27)
P1 + P2 = (13 , 6)
[2] P1 = (14 , 6)
```


Opérateurs arithmétiques sécurisés pour la cryptographie

Généralités :

- représentations des nombres et algorithmes arithmétiques efficaces
- représentations des nombres et algorithmes arithmétiques contre certaines attaques physiques
- développement coprocesseur ECC complet



Multiplication scalaire dans ECC

Opérandes : P un point de la courbe E, un entier $k = \sum_{i=0}^{n-1} k_i 2^i$

Résultat : le point $Q = [k]P = \underbrace{P + P + P + \ldots + P}_{k \text{ fois}}$

Algorithme classique : doublement-et-addition attaque SPA ! ! !

1 : $Q \longleftarrow P$
2 : **for** i **from** n-2 **to** 0 **do**
3 : $\quad Q \longleftarrow 2P$
4 : $\quad$ **if** $k_i = 1$ **then** $Q \longleftarrow Q + P$

Contre-mesures : recodages w-NAF ($n-1$ DBL et $\frac{n}{w+1}$ ADD)

$$
\begin{array}{rlllllllll}
k = 267 = & (\ 1 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 1\ )_2 \\
 & (\ 1 & 0 & 0 & 0 & 1 & 0 & \bar{1} & 0 & \bar{1}\ )_{2-NAF} \\
 & (\ 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 3\ )_{3-NAF} \\
 & (\ 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & \bar{5}\ )_{4-NAF} \\
 & (\ 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 11\ )_{5-NAF}
\end{array}
$$

Chaînes d'additions pour ECC

Implantation FPGA des chaînes d'additions proposées par Nicolas Méloni, collaboration UCC

Dans la multiplication scalaire $[k]P$, utiliser uniquement des additions de points de la courbe :

- robuste contre les attaques SPA
- $ADD(P_1, P_2) = (P_1 + P_2, P_1)$ avec P_1 et P_2 déjà calculés
- problème : trouver une petite chaîne

Exemple : chaînes d'additions pour $k = 113$

1	1	2	1	1	6	1	1	14	14	14	14	14	14	14	
1	2	3	5	6	7	13	14	15	29	43	57	71	85	99	113

1	1	1	1	4	5	5	14	14	19	47	
1	2	3	4	5	9	14	19	33	47	66	113

Recodage des nombres pour la sécurité : DBNS

DBNS : *Double-Base Number System*

$$x = \sum_{i=1}^{n} x_i 2^{a_i} 3^{b_i}, \text{ avec } x_i \in \{-1, 1\},\ a_i, b_i \geq 0$$

Exemple : $127 = 108 + 16 + 3 = 72 + 54 + 1 = \ldots$ (783 représentations)

	1	2	4	8	16
1					1
3	1				
9					
27			1		

	1	2	4	8
1	1			
3				
9				1
27		1		

Application : $k = 314159$, $[k]P =$

- $[2^4 3^9]P + [2^8 3^1]P - P$ — 12 DBL + 10 TPL + 2 ADD
- $3(3(3(3^3([2^4 3^3]P - P) - P) - P) - P) - P$ — 4 DBL + 9 TPL + 5 ADD

Source : L. Imbert

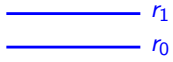
Codage des chiffres au niveau circuit

Codage classique pour un bit $b \in \{0, 1\}$:

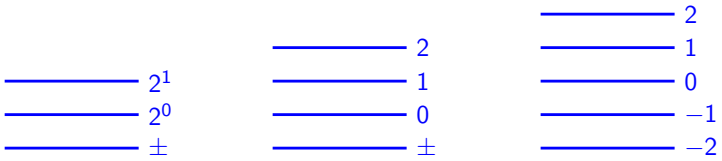
- $1 \Longrightarrow b = 1$
- $0 \Longrightarrow b = 0$

Codage double-rail d'un bit $b \in \{0, 1\}$:

- $(r_1, r_0) = (1, 0) \Longrightarrow b = 1$
- $(r_1, r_0) = (0, 1) \Longrightarrow b = 0$



Codages en base 4 de chiffres de $\{-2, -1, 0, 1, 2\}$:

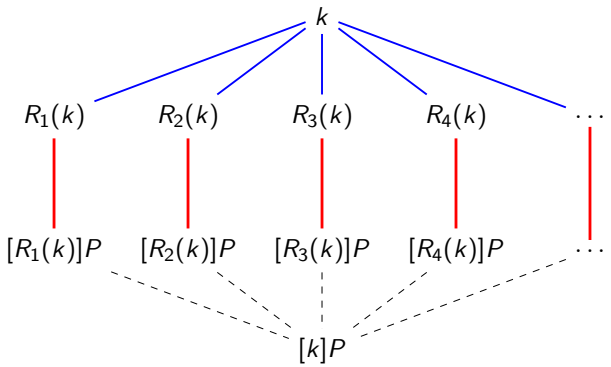


But : uniformisation du nombre de transitions (activité)
Surcoût : surface du circuit et éléments de mémorisation

Protection au niveau arithmétique

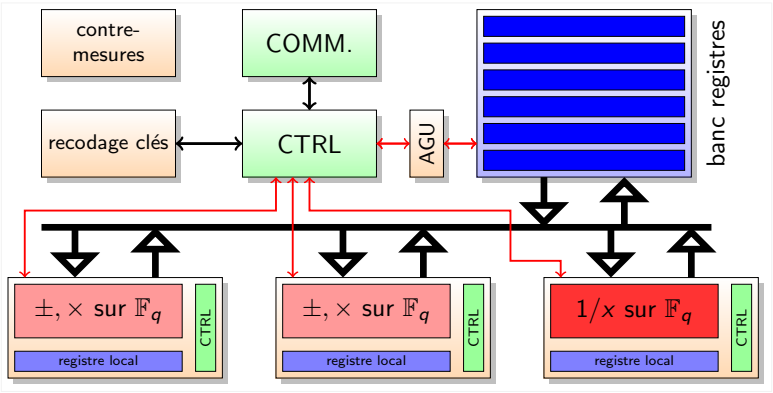
Un système redondant d'écriture des nombres, c'est :

- un moyen d'accélérer certaines opérations
- un moyen de représenter une même valeur de différentes façons



Proposition : utiliser des représentations redondantes aléatoires de k

Développement d'un coprocesseur ECC



- Unités fonct. : $\pm, \times, 1/x$ pour $\mathbb{F}_p$ et $\mathbb{F}_{2^m}$, recodage clés
- Mémoire : banc registres global + registres internes unités
- Contrôle : opérations (niveaux E et $\mathbb{F}_q$), ordon., paramètres

Conclusion

Travaux en arithmétique des ordinateurs :

- représentations des nombres
- algorithmes de calcul
- implantations matérielles et logicielles
- validation *a priori* et *a posteriori*
- modélisation et évaluation des performances
- outils d'aide à la conception d'opérateurs arithmétiques

Des liens avec :

- la conception de circuits
- l'architecture des machines
- le développement d'outils logiciels
  - bibliothèques de calculs
  - outils de CAO de circuits

Perspectives

Futur proche :

- avancées sur le coprocesseur ECC
  - opérateurs sécurisés
  - prototype FPGA
  - version en logiciel libre
- arithmétique par estimation (collab. U. Adelaide)
- gestion de la précision dans les outils de CAO (biblio. System C)

Futur moins proche :

- comparaison de la consommation des méthodes d'approximation de fonctions (tables, polynômes, additions-décalages, hybrides)
- arithmétiques pour circuits asynchrones
- reprendre les travaux sur FPOP
- opérateurs tolérants aux pannes et la variabilité de certains paramètres

Fin, des questions ?

Contact:

- `mailto:arnaud.tisserand@irisa.fr`
- `http://www.irisa.fr/prive/Arnaud.Tisserand/`
- Equipe-projet CAIRN `http://www.irisa.fr/cairn/`
- Laboratoire IRISA, CNRS–INRIA–Univ. Rennes 1
  6 rue Kérampont, BP 80518, F-22305 Lannion cedex, France

Merci