The PACE Library and Hardware Arithmetic Operators

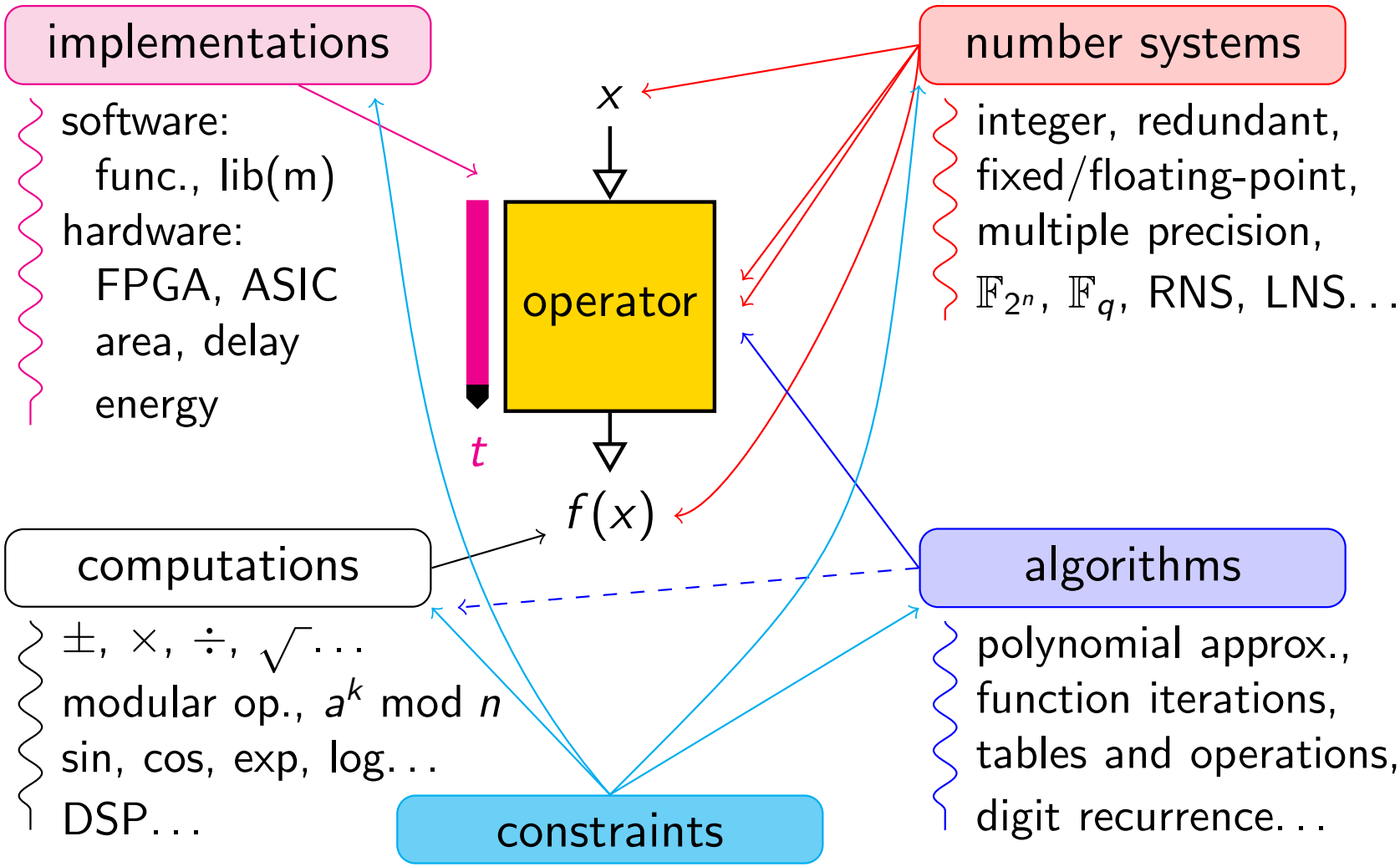
Arnaud Tisserand

LIRMM, CNRS–Univ. Montpellier 2
Arith Group

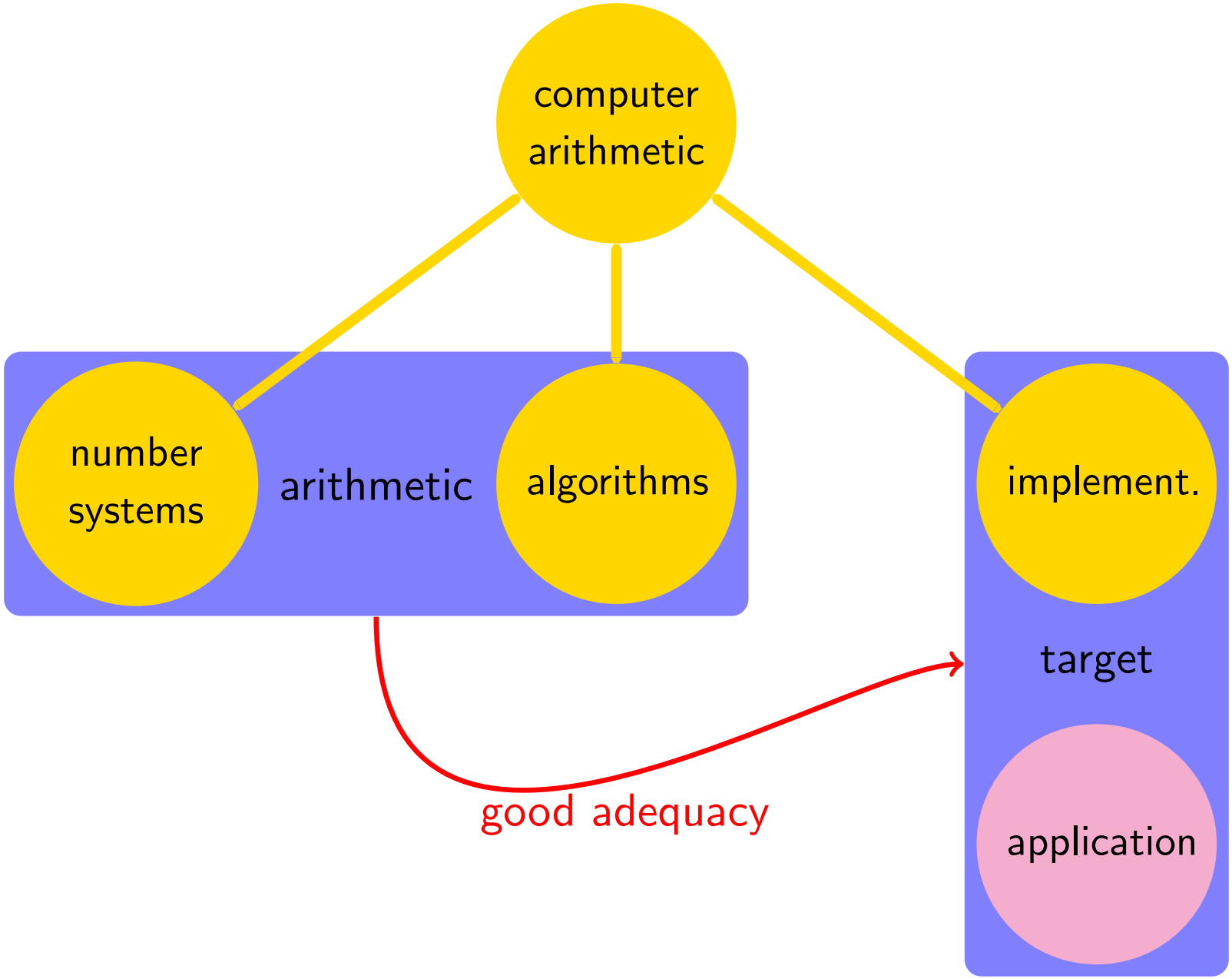
CryptArchi
Trégastel, June 1-4, 2008

Outline

Arithmetic Operator



Arithmetic(s) Optimization



Practical Problems in Computer Arithmetic

- limited support in design tools

  software: integer, floating-point, math. libraries
  hardware: integer, fixed-point, a few IP blocs

- validation

  verification of the correctness of a program (function, library, hardware bloc, circuit) at design time

- test

  verification of the correctness of an implementation

Our solutions:

- optimized and validated libraries (e.g. PACE)
- automatic generation of low-level descriptions (C and VHDL)
- include new arithmetic types and primitives in design tools (compilers, CAD tools)

Other Libraries/Tools

- GMP, arbitrary precision arithmetic on signed integers, rationals and floating point numbers
- NTL, data structures and algorithms for arbitrary length integers, and for vectors, matrices, and polynomials over the integers and over finite fields
- CLN, computations with all kind of numbers, including complex numbers, and univariate polynomials in arbitrary precision
- Miracl, crypto primitives (RSA, DH, ECC, AES, SHA2, etc)
- $\mathtt{mp}\mathbb{F}_q$, finite fields
- ZEN, arbitrary finite field arithmetic
- CAO, "an experimental cryptography-aware domain-specific language and associated compiler system"
- SAGE, open source mathematics software
- Maple, Magma and Pari/GP for validation

Some Research Activities in the ARITH Group

Computer arithmetic for cryptography applications:

- modular and finite field arithmetic
- implementation of basic crypto arithmetic primitives
- residue number systems (RNS)
- double-base number systems (DBNS)
- arithmetic library
- addition chains for ECC implementations
- secured arithmetic operators design
  - power-consumption aspects
  - fault injection
- implementations of applications
- hyperelliptic curves
- pairings

www.lirmm.fr/arith/

PACE Overview

Motivations:

- very limited mathematical support in languages and processors ("small" integers and floating-point approximation of real numbers)
- comparison of several solutions is difficult
  - hard to write all the solutions to test
  - needs a uniform test system for comparison accuracy
- fast validation of new algorithms/representations

Solution: PACE library for "Prototyping Arithmetic for Crypto Easily"

- C++
- templates
  - generic software
  - specialization for high performance (traits)
- LGPL license
- current version: ECC, prime fields, standard algorithms

PACE Team

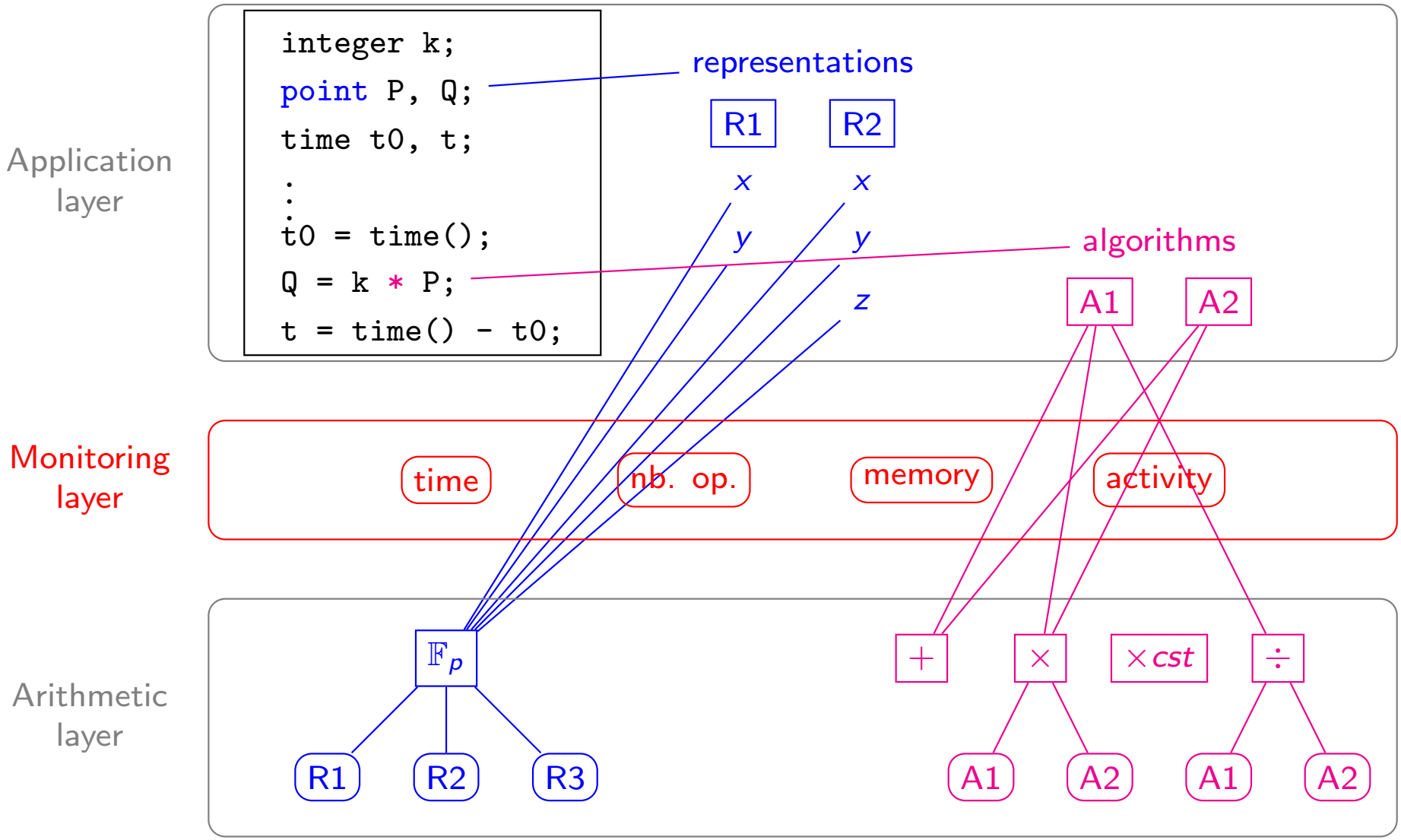
Main developers:

- Pascal Giorgi (associate professor Univ. Montpellier, LIRMM)
- Laurent Imbert (CNRS researcher, LIRMM)
- Arnaud Tisserand (CNRS researcher, LIRMM)

Contributors:

- Thomas Izard (master student Univ. Montpellier)
- Agostinho Peirera (research engineer Univ. Montpellier, 2007)

PACE Architecture



Arithmetic Layer

```
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;
```

produces

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

Arithmetic Layer

Current version:

- long integers
  - representations: GMP, basic, with guard bits (under dev.)
  - operations:
    $\pm, \times, {}^2, \div, \sqrt{}, {}^{-1}, mod, invmod, powmod, cmp, bit, popcount$...
- prime fields elements, $\mathbb{F}_P$ with general P
  - representations: GMP, basic, Montgomery (under dev.), with guard bits (under dev.)
  - operations: $\pm, \times, {}^2, {}^{-1}, =, bit, popcount$...

Future versions:

- $\mathbb{F}_P$ with specific P values
- $\mathbb{F}_2$ and extensions
- polynomials
- ...

Support for Multiple Representations and Algorithms

Goal:

- support multiple representations and algorithms
- very small modifications in the code
- compare several solutions

Method:

- generic programming: traits from C++
- abstract version vs. specialized versions
- default version (configuration)

Example:

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

Monitoring Layer

- number of operations (at each level)
- time:
  - system time
  - performance counters of the processor (PAPI and Perfctr)
- memory (#words, max. size)
- number of objects (max or current count)
- other values:
  - cache misses
  - Hamming weight (activity)

Very simple use of the monitors:

monitor(s) declaration
program bloc
read monitor(s) state(s) or trace(s)

Application Layer (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;
```

produces

```
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)
```

Validation

```
integer<100> p = 2003;
typedef gfp<100, p> fp_2003;
curve<fp_2003> E(1132,278);
typedef point_jac<fp_2003, E> poin
E.info();
point P1(1213, 408, 601);
point P2(1623, 504, 1559);
cout << "P1 = " << P1 << endl;
cout << "P2 = " << P2 << endl;
point P3 = P1 + P2;
assert(P3 == point(1683,1388));
cout << "P1 + P2 = " << P3 << endl
point P4 = 2 * P1;
assert(P4 == point(1467,143));
cout << "[2] P1 = " << P4 << endl;
point P5 = 763 * P1;
assert(P5 == point(1455,882));
cout << "[763] P1 = " << P5 << end
```

```
Elliptic curve defined by y^2 = x^
P1 = (1213 : 408 : 601)
P2 = (1623 : 504 : 1559)
P1 + P2 = (763 : 440 : 1934)
[2] P1 = (1800 : 1083 : 1684)
[763] P1 = (752 : 1146 : 543)
```

```
fp_2003 := FiniteField(2003);
E := EllipticCurve( [fp_2003 | 113
print E;
P1 := elt< E | 1120, 1391 >;
P2 := elt< E | 894, 1425 >;
print "P1 = ", P1;
print "P2 = ", P2;
P3 := P1 + P2;
print "P1 + P2 = ", P3;
P4 := 2 * P1;
print "[2] P1 = ", P4;
P5 := 763 * P1;
print "[763] P1 = ", P5;
```

```
Elliptic curve defined by y^2 = x^
P1 = (1120 : 1391 : 1)
P2 = (894 : 1425 : 1)
P1 + P2 = (1683 : 1388 : 1)
[2] P1 = (1467 : 143 : 1)
[763] P1 = (1455 : 882 : 1)
```

PACE and Hardware Arithmetic Operators

- Arithmetic blocs modeling/prototyping
  - math. level
  - bit level
  - advanced number systems and algorithms support
  - fast and simple modifications
  - monitoring (accurate op. count, activity estimation, resource requirements...)
- Validation support
  - design time verifications (size, math. properties, behavior...)
  - test vectors generation
- Links to[1]
  - arithmetic operator generators
  - timing support (scheduling)
  - architecture exploration

[1] Features under development inside and outside PACE

Future Prospects

Software:

- advanced algorithms
- $\mathbb{F}_P$ with specific P values
- $\mathbb{F}_2$ and extensions
- polynomials
- SCA protected algorithms for ECC
- RSA like crypto
- support/link to other tools/libraries

Hardware:

- operator generator
- SCA protected algorithms for ECC
- co-simulation

The end, some questions ?

Contact:

- mailto:arnaud.tisserand@lirmm.fr
- http://www.lirmm.fr/~tisseran
- Arith group
- LIRMM Laboratory, CNRS–Univ. Montpellier 2
  161 rue Ada. F-34392 Montpellier cedex 5. France

Thank you