

My 2 hours today:

1. Efficient arithmetic in finite fields
2. 10-minute break
3. Elliptic curves

My 2 hours tomorrow:

4. Efficient arithmetic on elliptic curves
5. 10-minute break
6. Choosing curves

# Efficient arithmetic in finite fields

D. J. Bernstein

University of Illinois at Chicago

Some examples of finite fields:

$$\mathbf{Z}/(2^{255} - 19).$$

$$(\mathbf{Z}/(2^{61} - 1))[t]/(t^5 - 3).$$

$$(\mathbf{Z}/223)[t]/(t^{37} - 2).$$

$$(\mathbf{Z}/2)[t]/(t^{283} - t^{12} - t^7 - t^5 - 1).$$

How quickly can we  
add, subtract, multiply  
in these fields?

Answer will depend on platform:  
AMD Athlon, Sun UltraSPARC IV,  
Intel 8051, Xilinx Spartan-3, etc.

Warning: different platforms  
often favor different fields!

## The first question

How to multiply big integers?

Child's answer: Use polynomial with coefficients in  $\{0, 1, \dots, 9\}$  to represent integer in radix 10.

With this representation, multiply integers in two steps:

1. Multiply polynomials.
2. "Carry" extra digits.

Polynomial multiplication involves *small* integers.

Have split one big multiplication into many small operations.

Example of representation:

$$839 = 8 \cdot 10^2 + 3 \cdot 10^1 + 9 \cdot 10^0 =$$

value (at  $t = 10$ ) of polynomial

$$8t^2 + 3t^1 + 9t^0.$$

Squaring:  $(8t^2 + 3t^1 + 9t^0)^2 =$

$$64t^4 + 48t^3 + 153t^2 + 54t^1 + 81t^0.$$

Carrying:

$$64t^4 + 48t^3 + 153t^2 + 54t^1 + 81t^0;$$

$$64t^4 + 48t^3 + 153t^2 + 62t^1 + 1t^0;$$

$$64t^4 + 48t^3 + 159t^2 + 2t^1 + 1t^0;$$

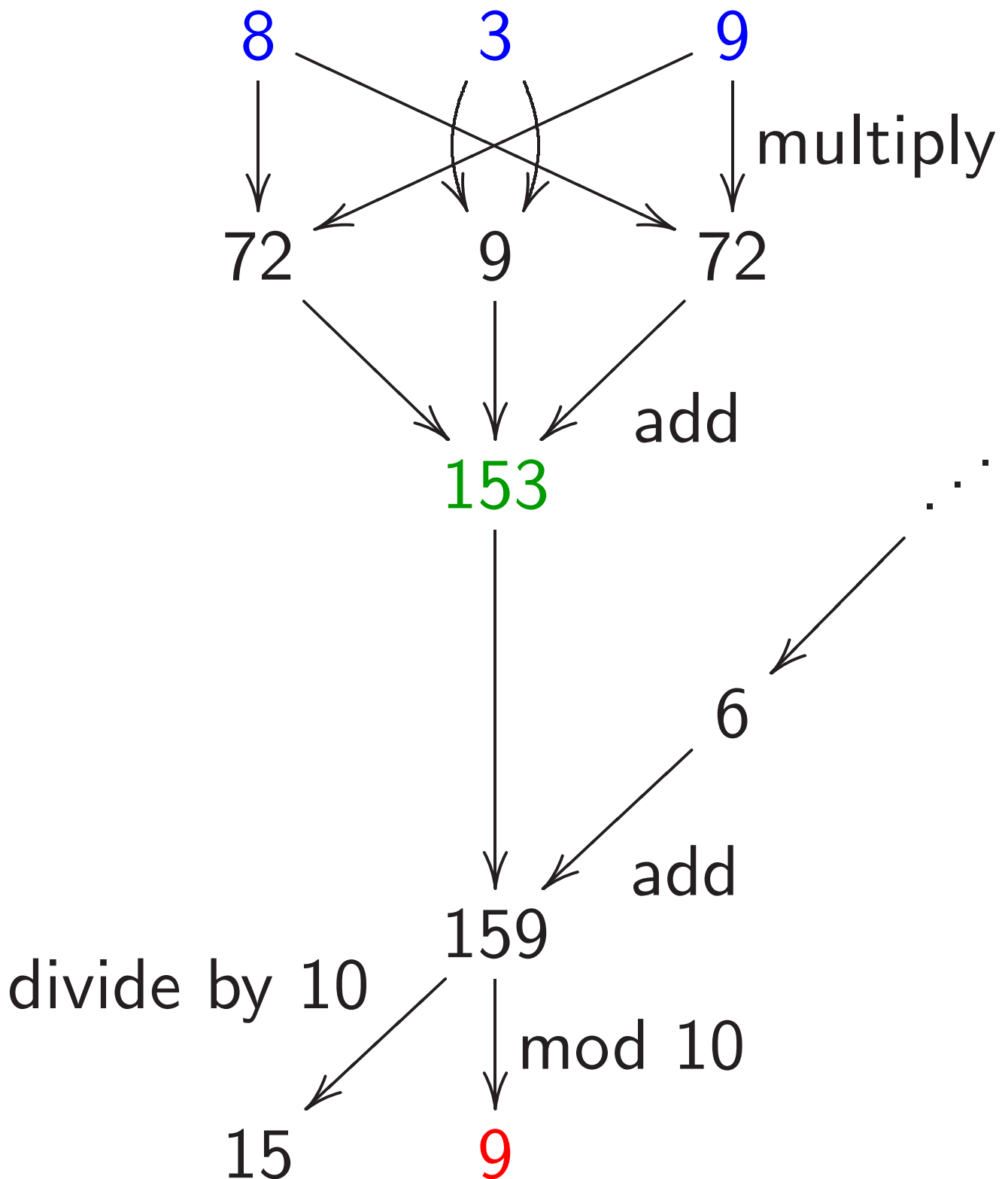
$$64t^4 + 63t^3 + 9t^2 + 2t^1 + 1t^0;$$

$$70t^4 + 3t^3 + 9t^2 + 2t^1 + 1t^0;$$

$$7t^5 + 0t^4 + 3t^3 + 9t^2 + 2t^1 + 1t^0.$$

In other words,  $839^2 = 703921$ .

What operations were used here?



Scaled variation:

$$839 = 800 + 30 + 9 =$$

value (at  $t = 1$ ) of polynomial

$$800t^2 + 30t^1 + 9t^0.$$

Squaring:  $(800t^2 + 30t^1 + 9t^0)^2 =$

$$640000t^4 + 480000t^3 + 153000t^2 + 540t^1 + 81t^0.$$

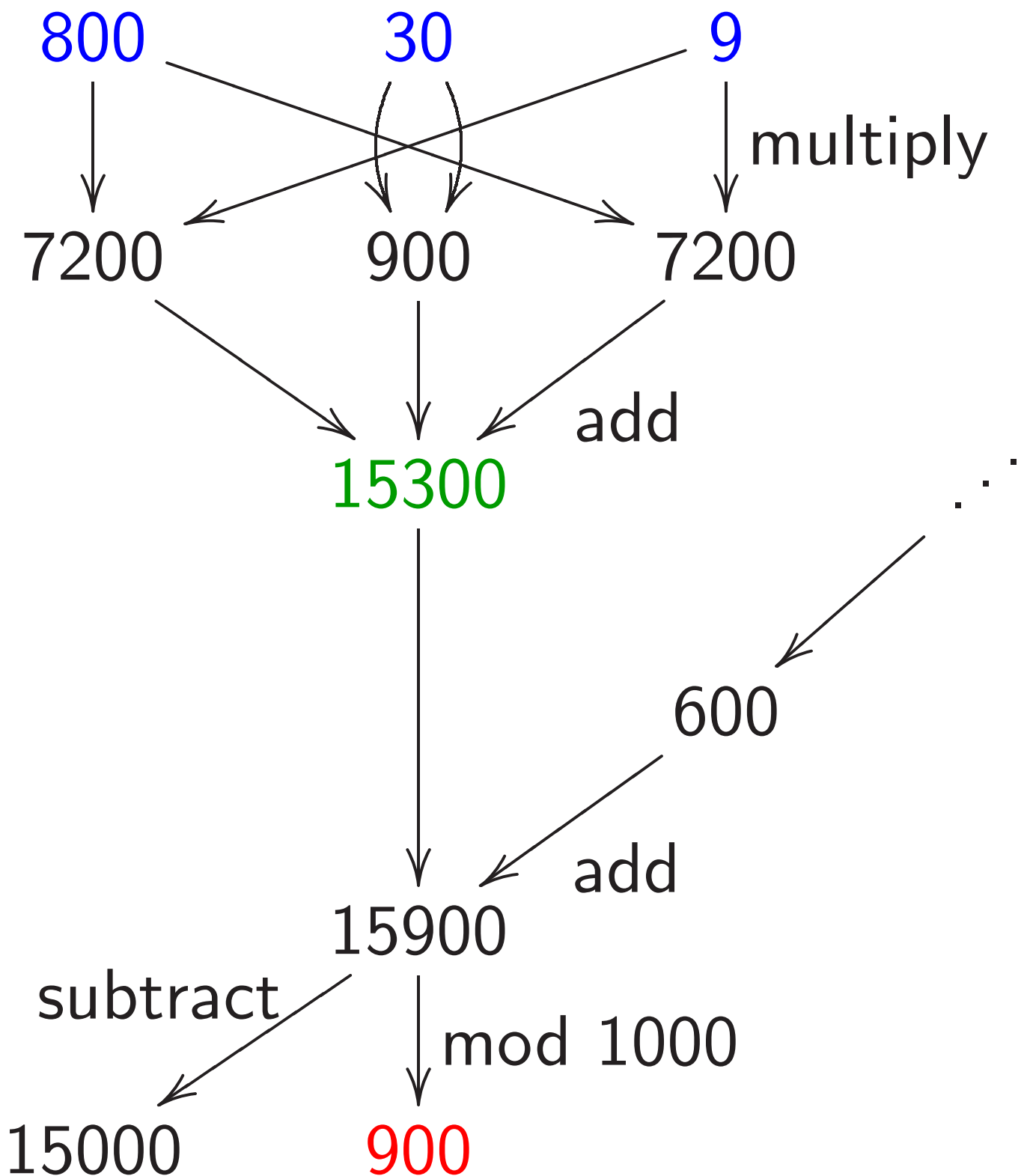
Carrying:

$$640000t^4 + 480000t^3 + 153000t^2 + 540t^1 + 81t^0;$$

$$640000t^4 + 480000t^3 + 153000t^2 + 620t^1 + 1t^0; \quad \dots$$

$$700000t^5 + 0t^4 + 30000t^3 + 900t^2 + 20t^1 + 1t^0.$$

What operations were used here?





## Speedup: double inside squaring

Squaring  $\dots + f_2 t^2 + f_1 t^1 + f_0 t^0$

produces coefficients such as

$$f_4 f_0 + f_3 f_1 + f_2 f_2 + f_1 f_3 + f_0 f_4.$$

Compute more efficiently as

$$2f_4 f_0 + 2f_3 f_1 + f_2 f_2.$$

Or, slightly faster,

$$2(f_4 f_0 + f_3 f_1) + f_2 f_2.$$

Or, slightly faster,

$$(2f_4) f_0 + (2f_3) f_1 + f_2 f_2$$

after precomputing  $2f_1, 2f_2, \dots$

Have eliminated  $\approx 1/2$  of the work  
if there are many coefficients.

## Speedup: allow negative coeffs

Recall  $159 \mapsto 15, 9$ .

Scaled:  $15900 \mapsto 15000, 900$ .

Alternative:  $159 \mapsto 16, -1$ .

Scaled:  $15900 \mapsto 16000, -100$ .

Use digits  $\{-5, -4, \dots, 4, 5\}$   
instead of  $\{0, 1, \dots, 9\}$ .

Several small advantages:

easily handle negative integers;

easily handle subtraction;

reduce products a bit.

## Speedup: delay carries

Computing (e.g.) big  $ab + c^2$ :  
multiply  $a, b$  polynomials, carry,  
square  $c$  poly, carry, add, carry.

e.g.  $a = 314, b = 271, c = 839$ :  
 $(3t^2 + 1t^1 + 4t^0)(2t^2 + 7t^1 + 1t^0) =$   
 $6t^4 + 23t^3 + 18t^2 + 29t^1 + 4t^0;$   
carry:  $8t^4 + 5t^3 + 0t^2 + 9t^1 + 4t^0.$

As before  $(8t^2 + 3t^1 + 9t^0)^2 =$   
 $64t^4 + 48t^3 + 153t^2 + 54t^1 + 81t^0;$   
 $7t^5 + 0t^4 + 3t^3 + 9t^2 + 2t^1 + 1t^0.$   
 $+: 7t^5 + 8t^4 + 8t^3 + 9t^2 + 11t^1 + 5t^0;$   
 $7t^5 + 8t^4 + 9t^3 + 0t^2 + 1t^1 + 5t^0.$

Faster: multiply  $a, b$  polynomials, square  $c$  polynomial, add, carry.

$$\begin{aligned} & (6t^4 + 23t^3 + 18t^2 + 29t^1 + 4t^0) + \\ & (64t^4 + 48t^3 + 153t^2 + 54t^1 + 81t^0) = \\ & 70t^4 + 71t^3 + 171t^2 + 83t^1 + 85t^0; \\ & 7t^5 + 8t^4 + 9t^3 + 0t^2 + 1t^1 + 5t^0. \end{aligned}$$

Eliminate intermediate carries.

Outweighs cost of handling slightly larger coefficients.

Important to carry between multiplications (and squarings) to reduce coefficient size; but carries are usually a bad idea for additions, subtractions, etc.

## Speedup: polynomial Karatsuba

Computing product of polys  $f$ ,  
with (e.g.)  $\deg f < 20$ ,  $\deg g < 20$ :  
400 coefficient mults,  
361 coefficient adds.

Faster: Write  $f$  as  $F_0 + F_1 t^{10}$   
with  $\deg F_0 < 10$ ,  $\deg F_1 < 10$ .  
Similarly write  $g$  as  $G_0 + G_1 t^{10}$ .

Then  $fg = (F_0 + F_1)(G_0 + G_1)t^{10}$   
 $+ (F_0 G_0 - F_1 G_1 t^{10})(1 - t^{10})$ .

20 adds for  $F_0 + F_1, G_0 + G_1$ .

300 mults for three products

$F_0G_0, F_1G_1, (F_0 + F_1)(G_0 + G_1)$ .

243 adds for those products.

9 adds for  $F_0G_0 - F_1G_1t^{10}$

with subs counted as adds

and with delayed negations.

19 adds for  $\dots (1 - t^{10})$ .

19 adds to finish.

Total 300 mults, 310 adds.

Larger coefficients, slight expense;  
still saves time.

Can apply idea recursively  
as poly degree grows.

Many other algebraic speedups  
in polynomial multiplication:  
Toom, FFT, etc.

Increasingly important as  
polynomial degree grows.

$O(n \lg n \lg \lg n)$  coeff operations  
to compute  $n$ -coeff product.

Useful for sizes of  $n$   
that occur in cryptography?  
Maybe; active research area.

## Using CPU's integer instructions

Replace radix 10 with, e.g.,  $2^{24}$ .

Power of 2 simplifies carries.

Adapt radix to platform.

e.g. Every 2 cycles, Athlon 64  
can compute a 128-bit product  
of two 64-bit integers.

(5-cycle latency; parallelize!)

Also low cost for 128-bit add.

Reasonable to use radix  $2^{60}$ .

Sum of many products of digits  
fits comfortably below  $2^{128}$ .

Be careful: analyze largest sum.



e.g. In 4 cycles, Intel 8051  
can compute a 16-bit product  
of two 8-bit integers.

Could use radix  $2^6$ .

Could use radix  $2^8$ ,  
with 24-bit sums.

e.g. Every 2 cycles, Pentium 4 F3  
can compute a 64-bit product  
of two 32-bit integers.

(11-cycle latency; yikes!)

Reasonable to use radix  $2^{28}$ .

Warning: Multiply instructions  
are very slow on some CPUs.

e.g. Pentium 4 F2: 10 cycles!

## Using floating-point instructions

Big CPUs have separate floating-point instructions, aimed at numerical simulation but useful for cryptography.

In my experience, floating-point instructions support faster multiplication (often much, much faster) than integer instructions, except on the Athlon 64.

Other advantages: portability; easily scaled coefficients.

e.g. Every 2 cycles, Pentium III can compute a 64-bit product of two floating-point numbers, and an independent 64-bit sum.

e.g. Every cycle, Athlon can compute a 64-bit product and an independent 64-bit sum.

e.g. Every cycle, UltraSPARC III can compute a 53-bit product and an independent 53-bit sum.

Reasonable to use radix  $2^{24}$ .

e.g. Pentium 4 can do the same using SSE2 instructions.

How to do carries in  
floating-point registers?

(No CPU carry instruction:  
not useful for simulations.)

Exploit floating-point rounding:  
add big constant,  
subtract same constant.

e.g. Given  $\alpha$  with  $|\alpha| \leq 2^{75}$ :  
compute 53-bit floating-point sum  
of  $\alpha$  and constant  $3 \cdot 2^{75}$ ,  
obtaining a multiple of  $2^{24}$ ;  
subtract  $3 \cdot 2^{75}$  from result,  
obtaining multiple of  $2^{24}$   
nearest  $\alpha$ ; subtract from  $\alpha$ .

## Reducing modulo a prime

Fix a prime  $p$ .

The prime field  $\mathbf{Z}/p$

is the set  $\{0, 1, 2, \dots, p - 1\}$

with  $-$  defined as  $- \bmod p$ ,

$+$  defined as  $+ \bmod p$ ,

$\cdot$  defined as  $\cdot \bmod p$ .

e.g.  $p = 1000003$ :

$$1000000 + 50 = 47 \text{ in } \mathbf{Z}/p;$$

$$-1 = 1000002 \text{ in } \mathbf{Z}/p;$$

$$117505 \cdot 23131 = 1 \text{ in } \mathbf{Z}/p.$$

# How to multiply in $\mathbf{Z}/p$ ?

Can use definition:

$$fg \bmod p = fg - p \lfloor fg/p \rfloor.$$

Can multiply  $fg$  by a precomputed  $1/p$  approximation; easily adjust to obtain  $\lfloor fg/p \rfloor$ .

Slight speedup: “2-adic inverse”; “Montgomery reduction.”

We can do better: normally  $p$  is chosen with a special form (or dividing a special form; see “redundant representations”) to make  $fg \bmod p$  much faster.

e.g. In  $\mathbf{Z}/1000003$ :

$$314159265358 =$$

$$314159 \cdot 1000000 + 265358 =$$

$$314159(-3) + 265358 =$$

$$-942477 + 265358 =$$

$$-677119.$$

Easily adjust to range

$$\{0, 1, \dots, p - 1\}$$

by adding/subtracting a few  $p$ 's.

(Beware timing attacks!)

Speedup: Delay the adjustment;

extra  $p$ 's won't damage

subsequent field operations.

Can delay carries until after multiplication by 3.

e.g. To square 314159

in  $\mathbf{Z}/1000003$ : Square poly

$$3t^5 + 1t^4 + 4t^3 + 1t^2 + 5t^1 + 9t^0,$$

obtaining  $9t^{10} + 6t^9 + 25t^8 +$

$$14t^7 + 48t^6 + 72t^5 + 59t^4 +$$

$$82t^3 + 43t^2 + 90t^1 + 81t^0.$$

Reduce: replace  $(c_i)t^{6+i}$  by

$$(-3c_i)t^i, \text{ obtaining } 72t^5 + 32t^4 +$$

$$64t^3 - 32t^2 + 48t^1 - 63t^0.$$

$$\text{Carry: } 8t^6 - 4t^5 - 2t^4 +$$

$$1t^3 + 2t^2 + 2t^1 - 3t^0.$$



To minimize poly degree,  
mix reduction and carrying,  
carrying the top sooner.

e.g. Start from square  $9t^{10} + 6t^9 + 25t^8 + 14t^7 + 48t^6 + 72t^5 + 59t^4 + 82t^3 + 43t^2 + 90t^1 + 81t^0$ .

Reduce  $t^{10} \rightarrow t^4$  and carry  $t^4 \rightarrow t^5 \rightarrow t^6$ :  $6t^9 + 25t^8 + 14t^7 + 56t^6 - 5t^5 + 2t^4 + 82t^3 + 43t^2 + 90t^1 + 81t^0$ .

Finish reduction:  $-5t^5 + 2t^4 + 64t^3 - 32t^2 + 48t^1 - 87t^0$ . Carry  $t^0 \rightarrow t^1 \rightarrow t^2 \rightarrow t^3 \rightarrow t^4 \rightarrow t^5$ :  
 $-4t^5 - 2t^4 + 1t^3 + 2t^2 - 1t^1 + 3t^0$ .

## Speedup: non-integer radix

Consider  $\mathbf{Z}/(2^{61} - 1)$ .

Five coeffs in radix  $2^{13}$ ?

$$f_4t^4 + f_3t^3 + f_2t^2 + f_1t^1 + f_0t^0.$$

Most coeffs could be  $2^{12}$ .

Square  $\dots + 2(f_4f_1 + f_3f_2)t^5 + \dots$ .

Coeff of  $t^5$  could be  $> 2^{25}$ .

Reduce:  $2^{65} = 2^4$  in  $\mathbf{Z}/(2^{61} - 1)$ ;

$$\dots + (2^5(f_4f_1 + f_3f_2) + f_0^2)t^0.$$

Coeff could be  $> 2^{29}$ .

Very little room for

additions, delayed carries, etc.

on 32-bit platforms.

Scaled: Evaluate at  $t = 1$ .

$f_4$  is multiple of  $2^{52}$ ;

$f_3$  is multiple of  $2^{39}$ ;

$f_2$  is multiple of  $2^{26}$ ;

$f_1$  is multiple of  $2^{13}$ ;

$f_0$  is multiple of  $2^0$ . Reduce:

$$\dots + (2^{-60}(f_4 f_1 + f_3 f_2) + f_0^2)t^0.$$

Better: Non-integer radix  $2^{12.2}$ .

$f_4$  is multiple of  $2^{49}$ ;

$f_3$  is multiple of  $2^{37}$ ;

$f_2$  is multiple of  $2^{25}$ ;

$f_1$  is multiple of  $2^{13}$ ;

$f_0$  is multiple of  $2^0$ .

Saves a few bits in coeffs.

## More finite fields

Fix a prime  $p$ . Fix a poly  $\varphi$  in one variable  $t$  with  $\varphi$  irreducible mod  $p$ .

The finite field  $(\mathbf{Z}/p)[t]/\varphi$  is the set of polynomials

$$f_{\deg \varphi - 1} t^{\deg \varphi - 1} + \cdots + f_1 t^1 + f_0 t^0$$

with each  $f_i \in \mathbf{Z}/p$

and with  $-, +, \cdot$  defined modulo  $p$  and modulo  $\varphi$ .

$(\mathbf{Z}/p)[t]/\varphi$  is an “extension” of the prime field  $\mathbf{Z}/p$ ; it has “characteristic”  $p$ .

e.g. 223 is prime, and poly  
 $t^6 - 3$  is irreducible mod 223,  
so  $(\mathbf{Z}/223)[t]/(t^6 - 3)$  is a field.

$223^6$  elements of field,  
namely polynomials  $f_5t^5 + f_4t^4 +$   
 $f_3t^3 + f_2t^2 + f_1t^1 + f_0t^0$   
with each  $f_i \in \{0, 1, \dots, 222\}$ .

After adding, subtracting,  
multiplying: replace  $t^6$  by 3,  
replace  $t^7$  by  $3t$ , etc.; and  
reduce coefficients modulo 223.

e.g.  $(9t^4 + 1)^2 = 81t^8 + 18t^4 + 1 =$   
 $243t^2 + 18t^4 + 1 = 18t^4 + 20t^2 + 1.$

Have two levels of polynomials  
when  $p$  is large: element  
of  $(\mathbf{Z}/p)[t]/\varphi$  is poly mod  $\varphi$ ;  
each poly coefficient is integer  
represented as poly in some radix.

e.g.  $f_4t^4 + f_3t^3 + f_2t^2 + f_1t^1 + f_0t^0$   
in  $(\mathbf{Z}/(2^{61} - 1))[t]/(t^5 - 3)$   
could have each coefficient  $f_i$   
represented as poly of degree  $< 3$   
in radix  $2^{61/3}$ .

When  $p$  is small, especially  $p = 2$ ,  
many speedups beyond this talk:  
batching coefficients,  
using fast Frobenius, et al.