Factoring integers with elliptic curves

("not more of the same")

D. J. Bernstein
University of Illinois at Chicago

GTEM Workshop on
"Computational Number Theory
and Arithmetic Geometry":
"computational aspects of
algebraic curves over finite fields"
etc.

Plausible conjecture:

ECM completely factors

every b-bit integer

in time $\leq 2^{b^{1/2+o(1)}}$.

"But that's obsolete!

NFS: $\leq 2^{b^{1/3+o(1)}}$."

Plausible conjecture:

ECM completely factors

every b-bit integer

in time $\leq 2^{b^{1/2+o(1)}}$.

"But that's obsolete!

NFS: $\leq 2^{b^{1/3+o(1)}}$."

ECM is still useful:

1. It finds small primes

much more quickly than NFS.

Plausible conjecture:

ECM completely factors

every b-bit integer

in time $\leq 2^{b^{1/2+o(1)}}$.

"But that's obsolete!

NFS: $\leq 2^{b^{1/3+o(1)}}$."

ECM is still useful:

1. It finds small primes

much more quickly than NFS.

2. It is an increasingly important

subroutine in NFS.

Plausible conjecture:

ECM completely factors

every b-bit integer

in time $\leq 2^{b^{1/2+o(1)}}$.

"But that's obsolete!

NFS: $\leq 2^{b^{1/3+o(1)}}$."

ECM is still useful:

1. It finds small primes

much more quickly than NFS.

2. It is an increasingly important

subroutine in NFS.

3. It is critical for *batch* NFS.

Let's look at NFS.

Sieving small integers $i > 0$
using primes $2, 3, 5, 7$:

1				
2	2			
3		3		
4	2	2		
5			5	
6	2		3	
7				7
8	2	2	2	
9			3	3
10	2			5
11				
12	2	2	3	
13				
14	2			7
15			3	5
16	2	2	2	2
17				
18	2		3	3
19				
20	2	2		5

etc.

Sieving i and $611 + i$ for small i using primes $2, 3, 5, 7$:

1	
2	2
3	3
4	2 2
5	5
6	2 3
7	7
8	2 2 2
9	3 3
10	2 5
11	
12	2 2 3
13	
14	2 7
15	3 5
16	2 2 2 2
17	
18	2 3 3
19	
20	2 2 5

612	2 2 3 3
613	
614	2
615	3 5
616	2 2 2 7
617	
618	2 3
619	
620	2 2 5
621	3 3 3
622	2
623	7
624	2 2 2 2 3
625	5 5 5 5
626	2
627	3
628	2 2
629	
630	2 3 3 5 7
631	

etc.

Have complete factorization of the "congruences" $i(611+i)$ for some i's.

$14 \cdot 625 = 2^1 3^0 5^4 7^1$.
$64 \cdot 675 = 2^6 3^3 5^2 7^0$.
$75 \cdot 686 = 2^1 3^1 5^2 7^3$.

$14 \cdot 64 \cdot 75 \cdot 625 \cdot 675 \cdot 686$
$= 2^8 3^4 5^8 7^4 = (2^4 3^2 5^4 7^2)^2$.

$\gcd\{611, 14 \cdot 64 \cdot 75 - 2^4 3^2 5^4 7^2\}$
$= 47$.

$611 = 47 \cdot 13$.

Why did this find a factor of 611?
Was it just blind luck:
$\gcd\{611, \text{random}\} = 47$?

No.

By construction 611 divides $s^2 - t^2$
where $s = 14 \cdot 64 \cdot 75$
and $t = 2^4 3^2 5^4 7^2$.
So each prime > 7 dividing 611
divides either $s - t$ or $s + t$.

Not terribly surprising
(but not guaranteed in advance!)
that one prime divided $s - t$
and the other divided $s + t$.

Why did the first three completely factored congruences have square product?
Was it just blind luck?

Yes. The exponent vectors $(1,0,4,1), (6,3,2,0), (1,1,2,3)$ happened to have sum 0 mod 2.

But we didn't need this luck!
Given long sequence of vectors, easily find nonempty subsequence with sum 0 mod 2.

This is linear algebra over $\mathbf{F}_2$.
Guaranteed to find subsequence
if number of vectors
exceeds length of each vector.

e.g. for $n = 671$:

$1(n+1) = 2^5 3^1 5^0 7^1$;
$4(n+4) = 2^2 3^3 5^2 7^0$;
$15(n+15) = 2^1 3^1 5^1 7^3$;
$49(n+49) = 2^4 3^2 5^1 7^2$;
$64(n+64) = 2^6 3^1 5^1 7^2$.

$\mathbf{F}_2$-kernel of exponent matrix is
gen by (0 1 0 1 1) and (1 0 1 1 0);
e.g., $1(n+1)15(n+15)49(n+49)$
is a square.

Plausible conjecture: $\mathbf{Q}$ sieve can separate the odd prime divisors of any n, not just 611.

Given n and parameter y:

Try to completely factor $i(n+i)$ for $i \in \{1, 2, 3, \ldots, y^2\}$ into products of primes $\leq y$.

Look for nonempty set of i's with $i(n+i)$ completely factored and with $\prod_i i(n+i)$ square.

Compute $\gcd\{n, s-t\}$ where $s = \prod_i i$ and $t = \sqrt{\prod_i i(n+i)}$.

Generalizing beyond $\mathbf{Q}$

The $\mathbf{Q}$ sieve is a special case of the number-field sieve (NFS).

Recall how the $\mathbf{Q}$ sieve factors 611:

Form a square as product of $i(i + 611j)$ for several pairs (i, j):
$14(625) \cdot 64(675) \cdot 75(686)$
$= 4410000^2$.

$\gcd\{611, 14 \cdot 64 \cdot 75 - 4410000\}$
$= 47$.

The $\mathbf{Q}(\sqrt{14})$ sieve
factors 611 as follows:

Form a square
as product of $(i + 25j)(i + \sqrt{14}j)$
for several pairs (i, j):

$$(-11 + 3 \cdot 25)(-11 + 3\sqrt{14})$$
$$\cdot (3 + 25)(3 + \sqrt{14})$$
$$= (112 - 16\sqrt{14})^2.$$

Compute
$s = (-11 + 3 \cdot 25) \cdot (3 + 25)$,
$t = 112 - 16 \cdot 25$,
$\gcd\{611, s - t\} = 13$.

Why does this work?

Answer: Have ring morphism $\mathbf{Z}[\sqrt{14}] \to \mathbf{Z}/611$, $\sqrt{14} \mapsto 25$, since $25^2 = 14$ in $\mathbf{Z}/611$.

Apply ring morphism to square:

$$(-11 + 3 \cdot 25)(-11 + 3 \cdot 25) \cdot (3 + 25)(3 + 25)$$
$$= (112 - 16 \cdot 25)^2 \text{ in } \mathbf{Z}/611.$$

i.e. $s^2 = t^2$ in $\mathbf{Z}/611$.

Unsurprising to find factor.

Generalize from $(x^2 - 14, 25)$
to (f, m) with irred $f \in \mathbf{Z}[x]$,
$m \in \mathbf{Z}$, $f(m) \in n\mathbf{Z}$.

Write $d = \deg f$,
$f = f_d x^d + \cdots + f_1 x^1 + f_0 x^0$.

Can take $f_d = 1$ for simplicity,
but larger f_d allows
better parameter selection.

Pick $\alpha \in \mathbf{C}$, root of f.
Then $f_d \alpha$ is a root of
monic $g = f_d^{d-1} f(x/f_d) \in \mathbf{Z}[x]$.

$$\mathbf{Q}(\alpha) \leftarrow \mathcal{O} \leftarrow \mathbf{Z}[f_d\alpha] \xrightarrow{f_d\alpha \mapsto f_d m} \mathbf{Z}/n$$

Build square in $\mathbf{Q}(\alpha)$ from congruences $(i - jm)(i - j\alpha)$ with $i\mathbf{Z} + j\mathbf{Z} = \mathbf{Z}$ and $j > 0$.

Could replace $i - jx$ by higher-deg irred in $\mathbf{Z}[x]$; quadratics seem fairly small for some number fields.
But let's not bother.

Say we have a square $\prod_{(i,j)\in S}(i - jm)(i - j\alpha)$ in $\mathbf{Q}(\alpha)$; now what?

$\prod(i - jm)(i - j\alpha)f_d^2$
is a square in $\mathcal{O}$,
ring of integers of $\mathbf{Q}(\alpha)$.

Multiply by $g'(f_d\alpha)^2$,
putting square root into $\mathbf{Z}[f_d\alpha]$:
compute r with $r^2 = g'(f_d\alpha)^2 \cdot$
$\prod(i - jm)(i - j\alpha)f_d^2$.

Then apply the ring morphism
$\varphi : \mathbf{Z}[f_d\alpha] \to \mathbf{Z}/n$ taking
$f_d\alpha$ to $f_d m$. Compute $\gcd\{n,$
$\varphi(r) - g'(f_d m)\prod(i - jm)f_d\}$.
In $\mathbf{Z}/n$ have $\varphi(r)^2 =$
$g'(f_d m)^2 \prod(i - jm)^2 f_d^2$.

How to find square product
of congruences $(i - jm)(i - j\alpha)$?

Start with congruences for,
e.g., y^2 pairs (i, j).

Look for y-smooth congruences:
y-smooth $i - jm$ and
y-smooth $f_d \operatorname{norm}(i - j\alpha) =$
$f_d i^d + \cdots + f_0 j^d = j^d f(i/j)$.
Here "y-smooth" means
"has no prime divisor $> y$."

Find enough smooth congruences.
Perform linear algebra on
exponent vectors mod 2.

Optimizing NFS

Finding smooth congruences
is *always* a bottleneck.

"What if it's much faster
than linear algebra?"
Answer: If it is, trivially
save time by decreasing y.

Optimizing NFS

Finding smooth congruences
is *always* a bottleneck.

"What if it's much faster
than linear algebra?"
Answer: If it is, trivially
save time by decreasing y.

Main job of NFS implementor:
speed up smoothness detection.

Optimizing NFS

Finding smooth congruences
is *always* a bottleneck.

"What if it's much faster
than linear algebra?"
Answer: If it is, trivially
save time by decreasing y.

Main job of NFS implementor:
speed up smoothness detection.

Other ways to make NFS fast:
optimize set of pairs (i, j),
choice of f, etc. Fun: e.g.,
compute $\int_{-\infty}^{\infty} \frac{dx}{((x-m)f)^{2/(d+1)}}$.

1977 Schroeppel "linear sieve," forerunner of QS and NFS:
Factor $n \approx s^2$ using congruences $(s+i)(s+j)((s+i)(s+j)-n)$.
Sieve these congruences.

1996 Pomerance:
"The time for doing this is unbelievably fast compared with trial dividing each candidate number to see if it is Y-smooth. If the length of the interval is N, the number of steps is only about $N \log\log Y$, or about $\log\log Y$ steps on average per candidate."

Fact: These simple "steps"
become very slow as y increases.
Distant RAM is very slow.

Sieving small primes isn't bad,
but sieving large primes is
much slower than arithmetic.

Every recent NFS record
actually uses other methods
to find large primes:
e.g., SQUFOF, $p-1$, ECM.

For optimized NFS for big n,
ECM is the most important
step in smoothness detection.

ECM speedup team:

1	2	3	4	Daniel J. Bernstein
1	2	3	4	Tanja Lange
1			4	Peter Birkner
1				Christiane Peters
	2	3		Chen-Mou Cheng
	2	3		Bo-Yin Yang
	2			Tien-Ren Chen
		3		Hsueh-Chung Chen
		3		Ming-Shing Chen
		3		Chun-Hung Hsiao
		3		Zong-Cing Lin

1. “ECM using Edwards curves.”
Prototype software: GMP-EECM.
New rewrite: EECM-MPFQ.

2. “ECM on graphics cards.”
Prototype CUDA-EECM.

3. “The billion-mulmod-per-second PC.”
Current CUDA-EECM,
plus fast mulmods on
Core 2, Phenom II, and Cell.

4. “Starfish on strike.”
Integrated into EECM-MPFQ.

5. Not covered in this talk:
early-abort ECM optimization.

Fewer mulmods per curve

Measurements of EECM-MPFQ for $B_1 = 1000000$:

$b = 1442099$ bits in
$s = \mathrm{lcm}\{1, 2, 3, 4, \ldots, B_1\}$.

$P \mapsto sP$ is computed using
1442085 ($= 0.99999b$) DBL +
98341 ($0.06819b$) ADD.

These DBLs and ADDs use
5112988**M** ($3.54552b$**M**) +
5768340**S** ($3.99996b$**S**) +
9635920**add** ($6.68187b$**add**).

Compare to GMP-ECM 6.2.3:

$P \mapsto sP$ is computed using
2001915 (1.38820b) DADD +
194155 (0.13463b) DBL.

These DADDs and DBLs use
8590140**M** (5.95669b**M**) +
4392140**S** (3.04566b**S**) +
12788124**add** (8.86772b**add**).

Compare to GMP-ECM 6.2.3:

$P \mapsto sP$ is computed using
2001915 (1.38820b) DADD +
194155 (0.13463b) DBL.

These DADDs and DBLs use
8590140**M** (5.95669b**M**) +
4392140**S** (3.04566b**S**) +
12788124**add** (8.86772b**add**).

Could do better! 0.13463b**M**
are actually 0.13463b**D**.
D: mult by curve constant.
Small curve, small P, ladder
$\Rightarrow 4b\mathbf{M} + 4b\mathbf{S} + 2b\mathbf{D} + 8b\mathbf{add}$.
EECM still wins.

HECM handles 2 curves using $2b\mathbf{M} + 6b\mathbf{S} + 8b\mathbf{D} + \cdots$ (1986 Chudnovsky–Chudnovsky, et al.); again EECM is better.

HECM handles 2 curves using
$2b\mathbf{M} + 6b\mathbf{S} + 8b\mathbf{D} + \cdots$
(1986 Chudnovsky–Chudnovsky, et al.); again EECM is better.

What about NFS? $B_1 = 587$?
Measurements of EECM-MPFQ:

$b = 839$ bits in s.

$P \mapsto sP$ is computed using
833 $(0.99285b)$ DBL $+$
131 $(0.15614b)$ ADD.

These DBLs and ADDs use
3552**M** $(4.23361b\mathbf{M}) +$
3332**S** $(3.97139b\mathbf{S}) +$
6308**add** $(7.51847b\mathbf{add})$.

Note: smaller window size
in addition chain,
so more ADDs per bit.

Compare to GMP-ECM 6.2.3:

Note: smaller window size
in addition chain,
so more ADDs per bit.

Compare to GMP-ECM 6.2.3:

$P \mapsto sP$ is computed using
4785**M** ($5.70322b$**M**) +
2495**S** ($2.97378b$**S**) +
7053**add** ($8.40644b$**add**).

Even for this small B_1,
EECM beats Montgomery ECM
in operation count.

Notes on current stage 2:

1. EECM-MPFQ jumps through the j's coprime to d_1.
GMP-ECM: coprime to 6.

2. EECM-MPFQ computes Dickson polynomial values using Bos–Coster addition chains.
GMP-ECM: ad-hoc, relying on arithmetic progression of j.

3. EECM-MPFQ doesn't bother converting to affine coordinates until the end of stage 2.

4. EECM-MPFQ uses NTL for poly arith in "big" stage 2.

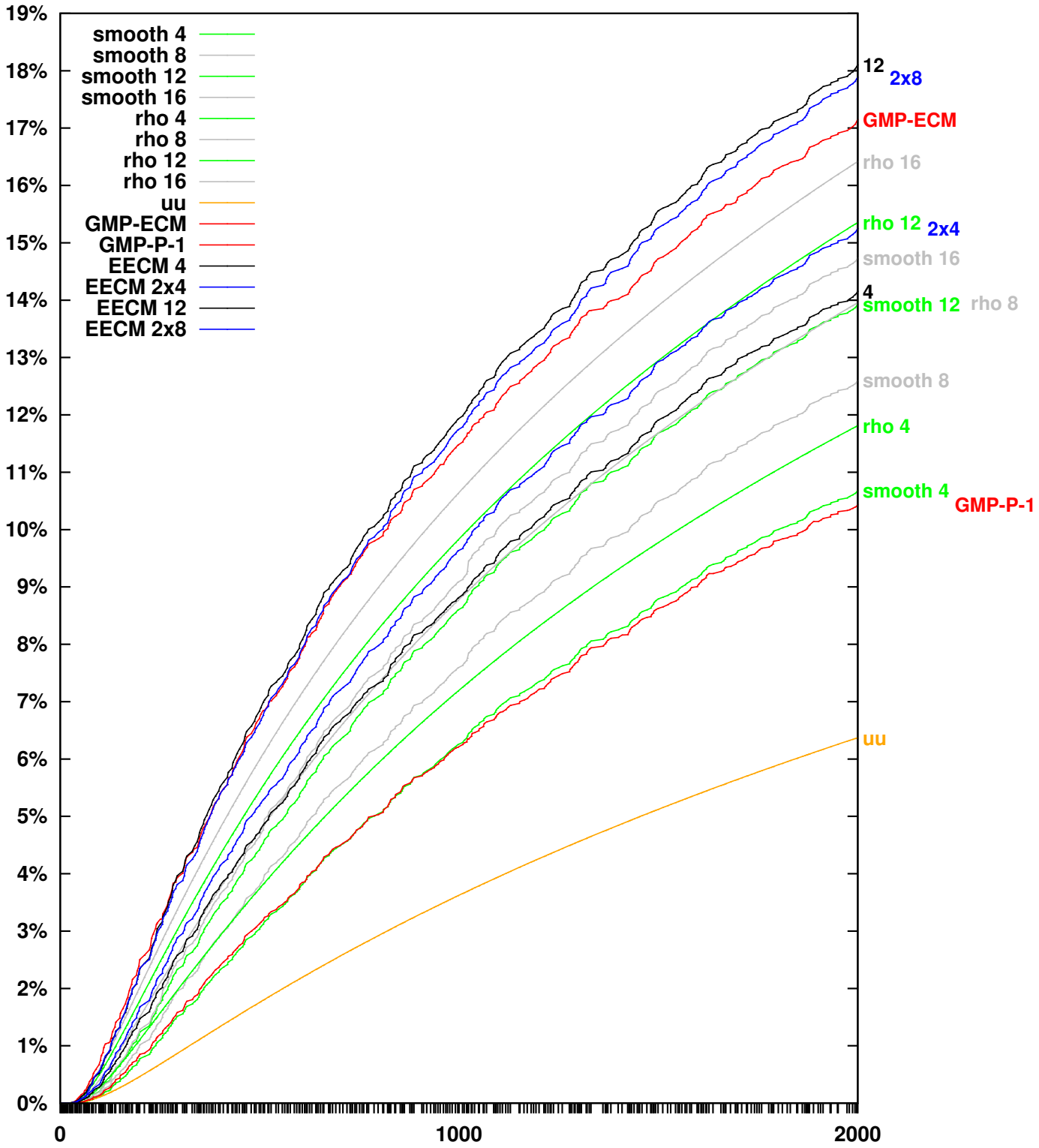
More primes per curve

1987/1992 Montgomery,
1993 Atkin–Morain
had suggested using torsion
$\mathbf{Z}/12$ or $(\mathbf{Z}/2) \times (\mathbf{Z}/8)$.

GMP-ECM went back to $\mathbf{Z}/6$.

"ECM using Edwards curves"
introduced new small curves
with $\mathbf{Z}/12$, $(\mathbf{Z}/2) \times (\mathbf{Z}/8)$.

Does big torsion really help?
Let's try a random sample
of 65536 30-bit primes.


19%
18%
17%
16%
15%
14%
13%
12%
11%
10%
9%
8%
7%
6%
5%
4%
3%
2%
1%
0%
0
1000
2000
smooth 4
smooth 8
smooth 12
smooth 16
rho 4
rho 8
rho 12
rho 16
uu
GMP-ECM
GMP-P-1
EECM 4
EECM 2x4
EECM 12
EECM 2x8
12
2x8
GMP-ECM
rho 16
rho 12
2x4
smooth 16
4
smooth 12
rho 8
smooth 8
rho 4
smooth 4
GMP-P-1
uu

Fastest known ADDs are for $-x^2 + y^2 = 1 + dx^2y^2$, which can't have > 8 torsion points.

"Starfish on strike":
Is the sacrifice in torsion justified by the ADD speedup?

Fastest known ADDs are for $-x^2 + y^2 = 1 + dx^2y^2$, which can't have > 8 torsion points.

"Starfish on strike":
Is the sacrifice in torsion justified by the ADD speedup?

Surprising phenomenon: $\mathbf{Z}/6$ $-x^2 + y^2 = 1 + dx^2y^2$ family finds *more* primes than $\mathbf{Z}/12$. Best ECM family known.

Fastest known ADDs are for $-x^2 + y^2 = 1 + dx^2y^2$, which can't have > 8 torsion points.

"Starfish on strike":
Is the sacrifice in torsion justified by the ADD speedup?

Surprising phenomenon: $\mathbf{Z}/6$ $-x^2 + y^2 = 1 + dx^2y^2$ family finds *more* primes than $\mathbf{Z}/12$. Best ECM family known.

Even more benefit from precomputing best curves.

Number of b-bit primes
found by 1000 different curves
$-x^2 + \cdots$ with $\mathbf{Z}/2 \times \mathbf{Z}/4$ torsion:

b	20	21	22
curve	$\frac{12}{343}, \frac{1404}{1421}$	$\frac{12}{343}, \frac{1404}{1421}$	$\frac{12}{343}, \frac{1404}{1421}$
#1	15486	22681	46150
curve	$\frac{27}{11}, \frac{5}{13}$	$\frac{27}{11}, \frac{5}{13}$	$\frac{27}{11}, \frac{5}{13}$
#2	14845	21745	43916
curve	$\frac{63}{20}, \frac{1}{244}$	$\frac{3}{14}, \frac{1}{17}$	$\frac{3}{14}, \frac{1}{17}$
#3	14537	21428	43482
#500	13706	19979	40993
#1000	13379	19475	40410

Number of b-bit primes
found by 1000 different curves
$x^2 + \cdots$ with $\mathbf{Z}/12$ torsion:

b	20	21	22
curve	...	...	...
#1	16276	23991	48076
curve	...	...	...
#2	16275	23970	48028
curve	...	...	...
#3	16273	23965	48020
#500	15977	23590	47521
#1000	15313	22714	45987

Number of b-bit primes
found by 1000 different curves
$-x^2 + \cdots$ with $\mathbf{Z}/6$ torsion:

b	20	21	22
curve	[932]	$\frac{825}{2752}, \frac{1521}{1504}$	$\frac{336}{527}, \frac{80}{67}$
#1	16328	24160	48424
curve	[94]	[982]	$\frac{825}{2752}, \frac{1521}{1504}$
#2	16289	24119	48378
curve	[785]	[265]	[306]
#3	16287	24113	48357
#500	16037	23735	47867
#1000	15399	22790	45828

Faster mulmods

ECM is bottlenecked by mulmods:

- practically all of stage 1;
- curve operations in stage 2 (pumped up by Dickson!);
- final product in stage 2, *except* fast poly arith.

GMP-ECM does mulmods with the GMP library.

. . . but GMP has slow API, so GMP-ECM has ≥ 20000 lines of new mulmod code.

```
$ wc -c<eecm-mpfq.tar.bz2
```

16031

Obviously EECM-MPFQ doesn't include new mulmod code!

```
$ wc -c<eecm-mpfq.tar.bz2
16031
```

Obviously EECM-MPFQ doesn't include new mulmod code!

MPFQ library (Gaudry–Thomé) does arithmetic in $\mathbf{Z}/n$ where number of n words is known at compile time.
Better API than GMP: most importantly, n in advance.

EECM-MPFQ uses MPFQ for essentially all mulmods.

GMP-ECM 6.2.3/GMP 4.3.2:

Tried 1000 curves, $B_1 = 2000$,
typical 240-bit n,
on 3.2GHz Phenom II x4.

Stage 1: $7.4 \cdot 10^6$ cycles/curve.

GMP-ECM 6.2.3/GMP 4.3.2:

Tried 1000 curves, $B_1 = 2000$,
typical 240-bit n,
on 3.2GHz Phenom II x4.

Stage 1: $7.4 \cdot 10^6$ cycles/curve.

EECM-MPFQ,
same 240-bit n, same CPU,
1000 curves, $B_1 = 2000$:
$5.2 \cdot 10^6$ cycles/curve.

Some speedup from Edwards;
some speedup from MPFQ.

What about stage 2?

GMP-ECM, 1000 curves,
$B_1 = 587$, $B_2 = 15366$,
Dickson polynomial degree 1:
$6.6 \cdot 10^6$ cycles/curve.
Degree 3: $9.5 \cdot 10^6$.

What about stage 2?

GMP-ECM, 1000 curves,
$B_1 = 587$, $B_2 = 15366$,
Dickson polynomial degree 1:
$6.6 \cdot 10^6$ cycles/curve.
Degree 3: $9.5 \cdot 10^6$.

EECM-MPFQ, 1000 curves,
$B_1 = 587$, $d_1 = 420$, range 20160
for primes $420i \pm j$:
$2.6 \cdot 10^6$ cycles/curve.
Degree 3: $3.1 \cdot 10^6$.

Summary: EECM-MPFQ uses fewer mulmods than GMP-ECM; takes less time than GMP-ECM; and finds more primes.

Summary: EECM-MPFQ uses fewer mulmods than GMP-ECM; takes less time than GMP-ECM; and finds more primes.

Are GMP-ECM and EECM-MPFQ fully exploiting the CPU? No!

Three recent efforts to speed up mulmods for ECM: Thorsten Kleinjung, for RSA-768; Alexander Kruppa, for CADO; and ours—see next slide.

Our latest mulmod speeds,
interleaving vector threads
with integer threads:

4×3GHz Phenom II 940:
$202 \cdot 10^6$ 192-bit mulmods/sec.

4×2.83GHz Core 2 Quad Q9550:
$114 \cdot 10^6$ 192-bit mulmods/sec.

6×3.2GHz Cell (Playstation 3):
$102 \cdot 10^6$ 195-bit mulmods/sec.

$500 GTX 295
is one card with two GPUs;
60 cores; 480 32-bit ALUs.
Runs at 1.242GHz.

Our latest CUDA-EECM speed:
$481 \cdot 10^6$ 210-bit mulmods/sec.

For $\approx$ $2000 can build PC
with one CPU and four GPUs:
$1300 \cdot 10^6$ 192-bit mulmods/sec.