CADO-NFS, a Number Field Sieve implementation

P. Gaudry[1], A. Kruppa[1], F. Morain[2],
L. Muller[1], E. Thomé[1], P. Zimmermann[1]

[1] CARAMEL/INRIA/LORIA ; [2] TANC/INRIA/LIX

Apr 27th, 2012

Plan

Motivations

Integer factorization ($N = pq \rightarrow$ find p, q) is a hard problem.

- Pre-1980's: a stumbling block in mathematical computations, and a challenging problem. Some significant advances in the 1970's.
- 1978-present: IF has attracted considerable attention because of its relevance for cryptography through the RSA cryptosystem.

Cado-nfs: an implementation of NFS

The fastest integer factoring algorithm is the Number Field Sieve.

- Very complicated algorithm. Embarks lots of number theory. (much more involved than, e.g., the ECM factoring algorithm)
- Very few available implementations. State of the art is at best bits and pieces from here and there.

Cado project. Write our own code. Joint effort, started in 2007.

- Actively developed. Playground for new ideas.
- Certainly beatable, but contains nice algorithms.
- No refrain to reorganizing the code to (changing) taste every so often.

Cado-nfs is LGPL, and written (almost) entirely in `C`. To date, ~ 120 kLOC.

Objectives for an NFS program

An NFS program like CADO-NFS can be used for various purposes.

- « below-NFS-threshold » numbers. Below 120dd, QS is faster.
  ⇒ intended for routine checking, timings are not the issue.
- Numbers which explore the limitations of the current code.
  Do growing sizes, add optimizations.
  Ongoing effort. Currently doing 700 bits.
- Record-size numbers. CADO-NFS can't factor `rsa768`, but participating to `rsa768` taught us a lot.

Note: CADO-NFS is clearly not an integrated factoring machinery. CADO-NFS does not include ECM, QS, ...

- No interaction with a user.
- Interface: a collection of programs driven by a main script.

Record sizes: crypto in sight

The feasibility limit explored by NFSrecords is used to determine key sizes for RSA.

- SSL/TLS. CA root certificates are installed by default in browsers.
  - Linux laptop, 2005: 1024b (50%), 2048b (48%), 4096b (2%) ;
  - Linux laptop, 2009: 1024b (31%), 2048b (58%), 4096b (10%).

- EMV credit cards (a.k.a. chip and pin).

  Most chip public keys are 960b. Some 1024b (until end of 2009, some had a 896b key).

Factoring experiments: decision-driving data for setting key sizes.

Plan

The GNFS setup

For factoring "general" N, GNFS uses:

- a number field $K = \mathbb{Q}(\alpha)$ defined by $f(\alpha) = 0$, for f irreducible over $\mathbb{Q}$ and $\deg f = d$;
- Another irreducible polynomial g such that f and g have a common root m mod N (example: $g = x - m$).

g defines the rational side, f defines the algebraic side.

Choosing f and g is referred to as the polynomial selection step.

General plan: Obtain relations, and combine them to obtain:

$$x^2 \equiv y^2 \mod N.$$

Relations in NFS

$$\begin{tikzcd}
 & \mathbb{Z}[x] \arrow[dl, "\psi^{(1)} : x \to m"'] \arrow[dr, "\psi^{(2)} : x \to \alpha"] & \\
\mathbb{Z}[m] \arrow[dr, "\varphi^{(1)} : t \to t \mod N"'] & & \mathbb{Z}[\alpha] \arrow[dl, "\varphi^{(2)} : \alpha \to m \mod N"] \\
 & \mathbb{Z}/N\mathbb{Z} &
\end{tikzcd}$$

Take for example $a - bx$ in $\mathbb{Z}[x]$. Suppose for a moment that:

- the integer $a - bm$ is smooth: product of factor base primes;
- the algebraic integer $a - b\alpha$ is also a product.

Then we have an multiplicative relation in $\mathbb{Z}/N\mathbb{Z}$. We can hope to combine many such relations to form a congruence of squares.

$$R = (a_1 - b_1 m) \times \cdots \times (a_k - b_k m) = \square,$$

$$A = (a_1 - b_1\alpha) \times \cdots \times (a_k - b_k\alpha) = \square,$$

$$\varphi^{(1)}(R) \equiv \varphi^{(2)}(R) \mod N.$$

Recognizing when $a - b\alpha$ factors

Major obstruction: $\mathbb{Z}[\alpha]$ not a UFD. “Factoring” $(a - b\alpha)$ won't work too well.

The proper object to look at is the factorization of the principal ideal generated by $(a - b\alpha)$ in the ring of integers of K.

- Some obstructions (ramifications, who's the maximal order) must be worked around.
- Essentially, we want the integer

$$\mathrm{Norm}_{K/\mathbb{Q}}(a - b\alpha) = \mathrm{Res}(a - bx, f) = b^d f(a/b) = F(a, b)$$

to be smooth. Nothing terribly complicated.

Complexity of NFS

For factoring an integer N, GNFS takes time:

$$L_N[1/3, (64/9)^{1/3}] = \exp\left((1 + o(1))(64/9)^{1/3}(\log N)^{1/3}(\log\log N)^{2/3}\right).$$

This is sub-exponential.

Note: some special numbers allow for a faster variant NFS, with complexity

$$L_N[1/3, (32/9)^{1/3}] = \exp\left((1 + o(1))(32/9)^{1/3}(\log N)^{1/3}(\log\log N)^{2/3}\right).$$

NFS: no panic

NFS might not be the simplest algorithm on earth, but:

- obstructions have been dealt with already long ago. See literature.
- the bottom line is simple: everything boils down to assembly/C/MPI.

Polynomial selection: find f, g;

Sieving: find many a, b s.t. $F(a, b) = b^d f(a/b)$ and $G(a, b)$ smooth.

Linear algebra: combine a, b pairs to get a congruence of squares.
(⇒ solve a large sparse linear system over $\mathbb{F}_2$.)

Square root: complete the factorization.

Recent progresses

Since RSA-155 (512 bits) in 1999, many improvements.

- Much better polynomial selection (Kleinjung, 2003, 2006).
- Very efficient sieving code (Franke, Kleinjung, 2003–).
- Very efficient cofactorization code (Kleinjung, Kruppa).

More recent state of the art, notably for linear algebra:

- Use block Wiedemann algorithm (BW), at separate locations.
- Use computer grids idle time to do linear algebra.
- Use sequences of unbalanced length in BW.

Plan

Polynomial selection

Asymptotic analysis of NFS gives formulae for:

- asymptotic optimal value for $\deg f$ (for an n-bit number).
- asymptotic optimal value for the coefficient sizes.

Trivial "base-m" approach:

- Choose the degree d. Choose an integer $m \approx N^{1/(d+1)}$;
- Write N in base m: $\qquad N = f_d m^d + f_{d-1} m^{d-1} + \cdots + f_0$.
- Pick $f = f_d X^d + \cdots + f_0$ and $g = X - m$.

We have an immense freedom in the choice of $m \Rightarrow$ can do better.

Polynomial selection algorithms

Algorithms aim at polynomial pairs (f, g) s.t. $F(a, b) = b^d f(a/b)$:

- is comparatively small over the sieving range.
- is often smooth (f with many roots mod small p).

Several relevant algorithms:

- Kleinjung (2006): handle an immense amount of possible polynomials, explore promising ones.
- Murphy (1999): rotation and root sieve: $(f, g) \rightsquigarrow (f + \lambda g, g)$.
- Kleinjung (2008): modification of the 2006 algorithm.

CADO-NFS has a `polyselect` program implementing this.

- polynomial root finding mod small p;
- knapsack-like problem solving;
- sieving for good λ; could use GPUs.

Plan

Sieving: a very old tool

In order to find (a, b) pairs for which $F(a, b)$ is smooth:

- For all small primes p (or prime powers) ;

Sieving: a very old tool

In order to find (a, b) pairs for which $F(a, b)$ is smooth:

- For all small primes p (or prime powers) ;
- for all roots r of f mod p, pick (a_0, b_0) s.t. $a_0 \equiv rb_0 \mod p$;

Sieving: a very old tool

In order to find (a, b) pairs for which $F(a, b)$ is smooth:

- For all small primes p (or prime powers) ;
- for all roots r of $f \bmod p$, pick (a_0, b_0) s.t. $a_0 \equiv rb_0 \mod p$;
- for all (u, v), mark $(a_0 + pu, b_0 + pv)$ as being divisible by p.

Keep (a, b) pairs which have been marked most.

Do this on both sides (f and g). Deciding in which order in subtle.

Note: NFS computation time is mostly spent on sieving.

Sieving: describing work

Lemma. For coprime (a, b),

- $\nu_p(F(a,b)) \geq 1$ iff $(a : b)$ is a zero of F in $\mathbb{P}^1(\mathbb{F}_p)$.
  Example: $f = 3x^2 + x + 1$.
  $F(a,b) = 3a^2 + ab + b^2 \equiv 0 \mod 3$ if either:
  - $(a : b) = (2 : 1)$ in $\mathbb{P}^1(\mathbb{F}_3)$: IOW, $a - 2b \equiv 0 \mod 3$.
  - $(a : b) = (1 : 0)$ in $\mathbb{P}^1(\mathbb{F}_3)$: IOW, $b \equiv 0 \mod 3$: "projective".
- More generally, (a, b)'s such that $\nu_p(F(a,b)) \geq k$ can be described as a set of points in $\mathbb{P}^1(\mathbb{Z}/p^\ell\mathbb{Z})$.

Starting point of sieving: compute the factor bases (both sides)

- Set of (p^ℓ, r), where $r < 2p^\ell$ encodes a point in $\mathbb{P}^1(\mathbb{Z}/p^\ell\mathbb{Z})$.
- Algebraic side harder than rational, but done offline anyway.
- root finding mod p;
- handle projective roots;
- handle powers. Some guaranteed headaches.

Typical problems with sieving

There are several practical shortcomings.

- The (a, b) space to be explored is large, but predicting in advance the yield for a range of (a, b) pairs is hard ;
- The yield drops as (a, b) grow ;
- $\Rightarrow$ diminishing returns.

Lattice sieving to the rescue.
Old idea (1993), but superiority demonstrated only after 2000.

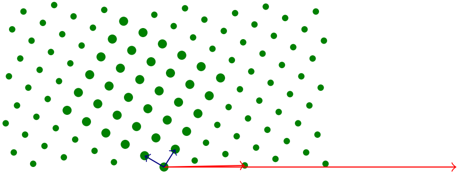
Lattice sieving

"special-$\mathfrak{q}$": prime ideal $\mathfrak{q} = \langle q, \alpha - r \rangle$.

How do we describe the set of pairs (a, b) such that $\mathfrak{q} \mid (a - b\alpha)$?
Answer: points in the lattice $\mathcal{L} = \langle e_0 = (r, 1),\ e_1 = (q, 0) \rangle$.

We would like to examine e.g. 2^{31} of these points. Which ones ?

- Bad idea: $\{(a, b) = ie_0 + je_1\}$ for $(i, j) \in [-2^{16}, 2^{16}[\times [0, 2^{15}[$.
  a gets then too large: $\approx q \times 2^{15}$.
- Better: reduced basis (e'_0, e'_1) and (i, j) in the same range.
  If the reduced basis is nice, we expect $a \approx b \approx 2^{16}\sqrt{q}$.



Lattice sieving

"special-$\mathfrak{q}$": prime ideal $\mathfrak{q} = \langle q, \alpha - r \rangle$.

How do we describe the set of pairs (a, b) such that $\mathfrak{q} \mid (a - b\alpha)$?
Answer: points in the lattice $\mathcal{L} = \langle e_0 = (r, 1),\ e_1 = (q, 0) \rangle$.

We would like to examine e.g. 2^{31} of these points. Which ones ?

- Bad idea: $\{(a, b) = ie_0 + je_1\}$ for $(i, j) \in [-2^{16}, 2^{16}[\times [0, 2^{15}[$.
  a gets then too large: $\approx q \times 2^{15}$.
- Better: reduced basis (e'_0, e'_1) and (i, j) in the same range.
  If the reduced basis is nice, we expect $a \approx b \approx 2^{16}\sqrt{q}$.

Benefits

- A factor of q is forced in the norm ;
- for q's of comparable size, we have comparable yields ;
- immense choice of special-$\mathfrak{q}$'s ;
- smaller sieve areas.

Lattice sieving: how do we sieve ?

Given a special-$\mathfrak{q}$ and $\begin{pmatrix} e'_0 \\ e'_1 \end{pmatrix} = \begin{pmatrix} a_0 & b_0 \\ a_1 & b_1 \end{pmatrix}$, we consider the lattice

$$\mathcal{L}_{\mathfrak{q}} = \left\{(a, b) = ie'_0 + je'_1\right\}.$$

All work is done on the (i, j) plane. A rectangle $\mathcal{R}_{(i,j)}$ is fixed.
The workplan for sieving for this special $\mathfrak{q}$ is:

- Describe locations to sieve in the (i, j) plane.
- Sieve "small" factor base primes.
- Sieve "large" factor base primes.
- Do this for both sides.
- Locations which have been marked most need to be factored.

Sieve locations in the (i, j) plane.

Let p be a prime (power) coprime to $\mathfrak{q}$. We have a homography:

$$h_{\mathfrak{q}} : \begin{cases} \mathbb{P}^1(\mathbb{Z}/p\mathbb{Z}) & \rightarrow & \mathbb{P}^1(\mathbb{Z}/p\mathbb{Z}), \\ (i : j) & \mapsto & (a : b) = (ia_0 + ja_1 : ib_0 + jb_1). \end{cases}$$

Starting from a description S_p of the (a, b) sieve locations:

$$\begin{aligned} \{(i,j), p \mid F(a,b)\} &= \{(i,j), (a : b) \in S_p \subset \mathbb{P}^1(\mathbb{Z}/p\mathbb{Z})\}, \\ &= \{(i,j), h_{\mathfrak{q}}(i : j) \in S_p\}, \\ &= \{(i,j), (i : j) \in h_{\mathfrak{q}}^{-1} S_p\}. \end{aligned}$$

- This change of basis must be redone for each $\mathfrak{q}$.
- relatively cheap because independent of the sieve area size.
- Need to compute inverses modulo factor base primes.

Fine points of sieving

For a given $\mathfrak{q}$, explore some $\mathcal{R}_{(i,j)}$ of size e.g. 2^{31}.

- Divide into areas matching L1 cache size (64kb typically), to be processed one by one.
- Small primes hit often: once per row.
- Larger primes hit rarely. Rather maintain a "schedule" list to circumvent cache misses: "bucket sieving".
- Use multithreading.

CADO-NFS implements this in `las`.

- Hot spots in assembly; Use vector instructions when relevant;
- Optimize some data structures to reduce memory footprint;
- Strive to eliminate badly predictable branches;
- POSIX threads;
- Factoring good (a, b)'s: Use $p \pm 1$ and special-purpose ECM.

Plan

Fast forward

The output of the sieve process is a set of relations.

These undergo:

- Filtering: making a small relation set from a large one ;
- After filtering, linear system solving.

Algorithmically, nothing very new in filtering since Cavallar (2000).

Implementation in Cado-nfs:

- Hash tables all over the place;
- Minimum spanning trees to help decision;
- Has supported MPI distribution at some point;

Does the job so far.

Linear algebra

Must combine relations so that they consist of only squares.

This rewrites as a linear system. (everything reduces to lin. alg. !)

- matrix M: a relation appears in each row. Coefficients are multiplicities of prime factors (and ideals). Most are zero.
- A vector v such that

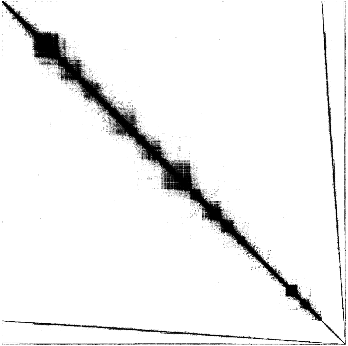
  $$vM = 0 \mod 2$$

  indicates which relations to combine in order to obtain only squares (even multiplicities).

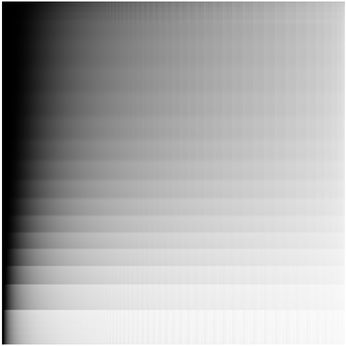
Equivalently, we rephrase this as a linear system $Mv = 0$ (transposing M).

Note: linear algebra mod 2 differs much from linear algebra over $\mathbb{C}$.

$\mathbb{F}_2$ is exact, and positive characteristic



(some PDE example)



(a factoring matrix)

Linear algebra

We have an $N \times N$ matrix M. We want to solve $Mw = 0$.

The matrix M is large, (very) sparse, and defined over $\mathbb{F}_2$.

Because of sparsity, we want a black box algorithm.

$$v \longrightarrow \boxed{M} \longrightarrow M \times v$$

There are several sparse linear algebra algorithms suitable for $\mathbb{F}_2$:

- Lanczos ;
- Wiedemann ; others.

These early suggestions are unsuitable. Bit arithmetic: slow. Also, failure probability $1/\#\mathbb{F}_2 = 1/2$ is not so tempting...

Block algorithms

Block algorithms apply the black box to e.g. $n = 64$ vectors at a time. (n is prescribed by the hardware)

- Block Lanczos (BL). $\frac{2N}{n-0.76}$ black box applications ;
- Block Wiedemann (BW). $\frac{3N}{nn'}$, n' times (n' small).

BL is appealing if one has a large cluster.

BW is preferred since it offers distribution opportunities.

Block Wiedemann: workplan

- Initial setup. Choose starting blocks of vectors x and y.
- Sequence computation. Want L first terms of the sequence:

$$a_i = x^T M^k y.$$

  - Computing one term after another, this boils down to our black box $v \mapsto Mv$.
  - This computation can be split into several independent parts (which all know M).
- Compute some sort of minimal polynomial.
- Build solution as:

$$v = \sum_{k=0}^{\deg f} M^k y f_k.$$

  - Again, this uses the black box.
  - Can be split into many independent parts (which all know M).

Linear algebra: size matters

The matrix M itself is soon out of reach for core storage.

- 2005: kilobit SNFS: 64M rows/cols, 10G non-zero coeffs. About 30GB.
- 2010: 768b GNFS: 192M rows/cols, 27G non-zero coeffs. About 75GB.

Computing $M \times v$ is also a lot of work. Try to use many processors if possible.

This is a classical HPC concern.

- Split the matrix into equal parts.
- Exploit high-bandwith channels: shared memory, infiniband network.

Features of the CADO-NFS BW code

CADO-NFS has a complete BW implementation.

Sequence computation:

- POSIX threads;
- MPI – implementation agnostic. Some optimized collectives;
- Some kind of "sparse binary BLAS" used. Assembly;
- (Stem of) capability to switch to other base field;
- Mostly `C`, some `C++`. Wrapper script in Perl.

Minimal polynomial computation using a quasi-linear algorithm.

- recursive structure;
- arithmetic on matrices of polynomials over $\mathbb{F}_2$.
- very old code, needs rework.

Plan

The square root step

Our congruence of squares actually comes as:

$$(a_1 - b_1 m) \times \cdots \times (a_k - b_k m) \equiv \phi((a_1 - b_1 \alpha) \times \cdots \times (a_k - b_k \alpha)) \mod N.$$

- Both sides are known to factor with even multiplicities: they are squares.
- BUT computing the square root is in fact non trivial (esp. on algebraic side).

CADO-NFS implements quasi-linear algorithms for this

- Newton lifting.
- Arithmetic modulo fixed degree polynomials.
- Suitable for current records.
- Alternative algorithm (waives a number theoretic assumption):
  - Explicit CRT.
  - Can be distributed with MPI.

There exists a more advanced square root algorithm for this step (Montgomery), but it needs more software support.

Plan

Conclusion and further work

Many points would be interesting to improve.

- Polyselect with GPUs (but `msieve` does this already).
- Lattice siever needs cleanup, and some obvious improvements.
- Filtering currently can't handle record sizes.
- Linear algebra sparse BLAS can be improved.
- Linear algebra minimal polynomial step must be reworked.
- The whole chain could be adapted to discrete log computation.