Factoring into large primes with P-1, P+1 and ECM

Alexander Kruppa

LORIA

CADO workshop Nancy, 9. October 2008

Using P-1, P+1 and ECM in NFS

- For large factoring projects, memory becomes a limiting resource. Large factor base bound B requires large sieve region size S to maintain $O(S \log\log B)$ complexity

- Fit larger project in available memory, use alternatives to sieving: smaller factor base, allow larger residual to survive sieving, find large primes $> B$ by other methods

- We investigate the efficiency of the P-1, P+1 and Elliptic Curve methods for this task. How fast can their implementation be for small input? How should they be combined for finding the large primes?

Tuning the sieving for many large primes

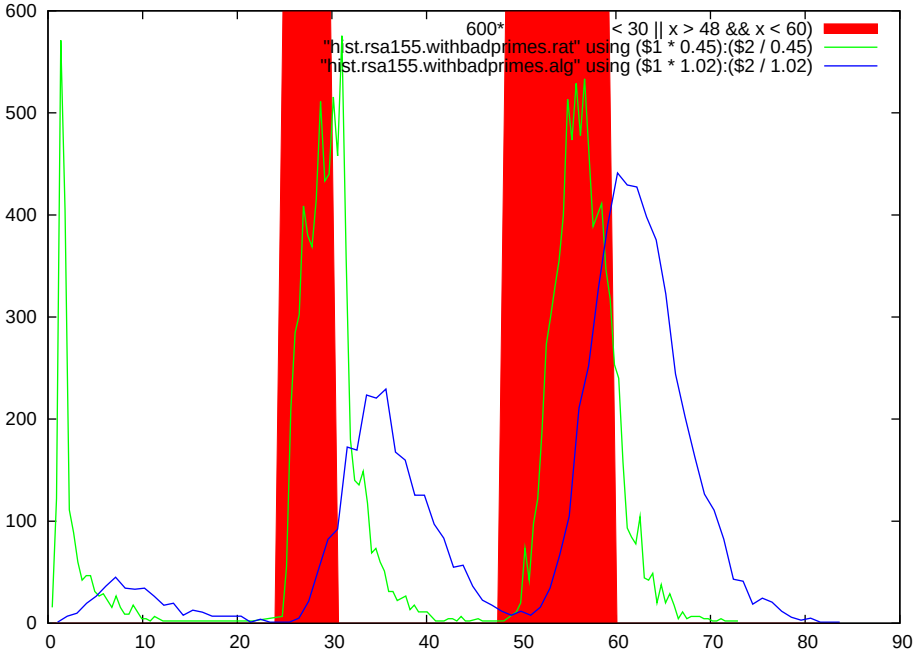
- Folklore: sieving doesn't have to be too accurate. True if number of survivors is small either way

- With many large primes, number of survivors explodes, and so does refactoring time. Example: RSA155 with current CADO lattice siever, 2 large primes on each side $(S = 2^{25}, B = 2^{24}, L = 2^{30}, \lambda_a = 2.4, \lambda_r = 2.2)$: 6496 survivors, 3 large primes on each side $(\lambda_a = 3.2, \lambda_r = 3.0)$: 282381 survivors

- We need to: sieve more accurately to reduce number of survivors, make refactoring deal with many survivors efficiently

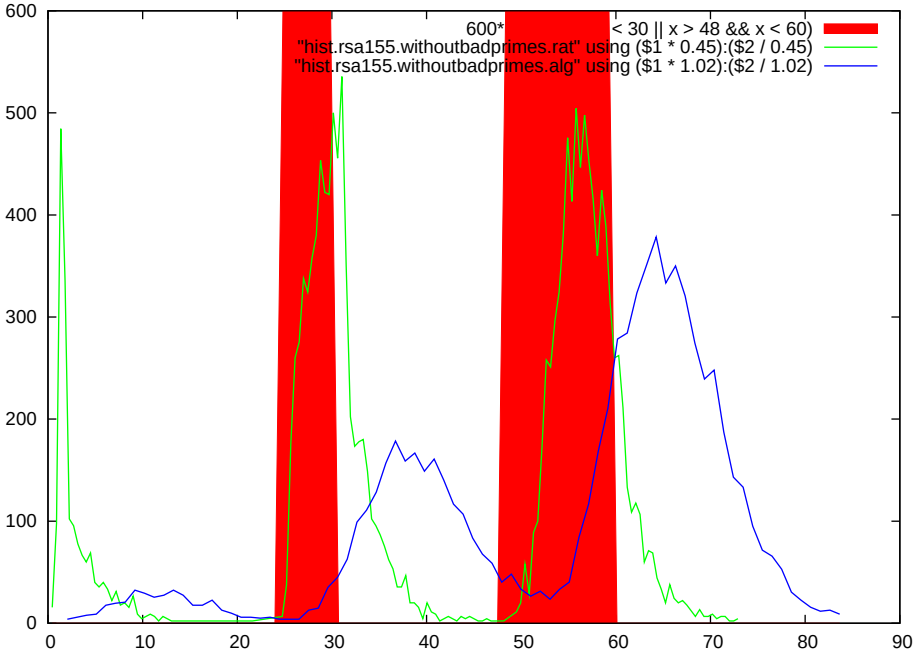
Sieving accuracy

- Sieving small primes, bad primes, prime powers allows for smaller lambda. In CADO siever we don't sieve prime powers yet (TBD)
- Accurate enough sieving would allow discarding reports for cofactors c in the "forbidden zone" $L < c < B^2, L^2 < c < B^3, \ldots$



Sieving accuracy (cont.)

- Comparison: sieving without bad primes



Trial dividing small primes

- With n survivors, trial dividing a prime with r roots takes $O(n)$ while resieving takes $O(rS/p)$. Trial divide those p with $p < cr\frac{S}{n}$ for some c.

- Trial division: Word size w, e.g. $w = 2^{64}$. For a given $p < \sqrt{w/l}$, precompute $w \bmod p$, $w^2 \bmod p, \ldots, w^l \bmod p$ (Montgomery). Also precompute $p_{\text{inv}} = p^{-1} \pmod w$ and $p_{\text{lim}} = \left\lfloor \frac{w-1}{p} \right\rfloor$.

- For input $n = \sum_{i=0}^{l} n_i w^i$, compute $r_1 w + r_0 = \sum_{i=0}^{l} n_i (w^i \bmod p)$, so that $r_1, r_0 < w$. Compute $s_1 w + s_0 = r_1(w \bmod p) + r_0$, now $s_1 \leq 1, s_0 < w$. Finally $t = s_1(w \bmod p) + s_0 < w$.

- Divisibility test: $p \mid n$ iff $(t \cdot p_{\text{inv}}) \bmod w \leq p_{\text{lim}}$. Under multiplication by $p_{\text{inv}} \bmod w$, multiples of p map to $[0, p_{\text{lim}}]$, rest to $]p_{\text{lim}}, w[$

Parameters for P-1, P+1

- For P-1, $x_0 = 2$. Fast left-to-right exponentiation, 2 is quadratic residue for $p \equiv 1 \pmod 8$
- For P+1, $x_0 = 2/7$. We get group order $p+1$ if $\left(\frac{\Delta}{p}\right) = -1$, $\Delta = x_0^2 - 4$, and order $p-1$ otherwise. With $x_0 = 2/7$, $\Delta = -192/49 = -3\left(\frac{8}{7}\right)^2$, so $p-1$ for $p \equiv 1 \pmod 6$ and $p+1$ for $p \equiv 5 \pmod 6$: order always divisible by 6
- So good choice of x_0 gives us order $p+1$ still for only half of the primes, but for the half where $p+1$ is more likely smooth than $p-1$
- Significant effect: of the 480831 primes in $[2^{30}, 2^{30} + 10^7]$,
  P-1 with $B_1 = 315, B_2 = 3000$ finds 36729,
  P+1 with $x_0 = 2/7$ finds 46726, 27% more

Parameters for ECM

- Two curve parameterizations implemented: by Brent-Suyama and by Montgomery (torsion 12)
- Anomaly: Brent-Suyama with $\sigma = 11$ finds more factors than other curves. With $B_1 = 250$ and $B_2 = 10000$, $\sigma = 11$ finds 7% more primes $\sim 2^{30}$ than $\sigma = 10$
- With $\sigma = 11$: average exponent of 2 in group order: $\approx 11/3$. With other sigmas: $\approx 10/3$. Exponents of other primes seem unchanged. Reason not clear, guess: some roots of small division polynomials having smaller algebraic degree? TBD
- Montgomery torsion 12 curves all seem to have average exponent $\approx 11/3$ of 2, but more expensive to initialise
- TBD: find a few good, cheap to initialise curves

Arithmetic for small moduli

- Modular arithmetic implemented as inline functions in C header files
- Currently arithmetic for 1 and 1.5 words ($\leq$64 and $\leq$96 bit moduli on 64 bit machines), 2 and 3 words TBD. Modulo reduction with REDC
- Implementation of factoring algorithms largely independent of modular arithmetic, #include-ing different headers produces factoring code for different input sizes
- Compiler does reasonably good job inlining functions, some speedup possible by writing e.g. elliptic curve addition in assembly for different modulus sizes

P+1, ECM stage 1

- P+1 uses Chebyshev polynomials, ECM uses curves in projective coordinates (Montgomery form): Lucas chains for addition (to compute $a + b$, $a - b$ must be known)

- For given B_1, Lucas chain for stage 1 is precomputed. Uses PRAC for generating chains for primes $p < B_1$. Usually makes optimal chains (not for $p = 421, 751, 1087, 1201, \ldots$ rare enough)

- PRAC uses 9 rules to generate Lucas chains. Bytecode stores the sequence of rules to apply. Sequence highly repetitive: uses compression (static dictionary) to combine frequent sequences of rules into one code: less parsing overhead (P+1)

- P+1, ECM stage 1 parses the bytecode (`switch` statement implementing arithmetic for the different rules)

P-1, P+1 stage 2

- P-1, P+1 use common enhanced standard stage 2. For P-1 stage 1 output x, compute $X = x + x^{-1}$. For P+1, stage 1 output is $X = \alpha + \alpha^{-1}$, where $\alpha^2 - x_0\alpha + 1 \equiv 0 \pmod N$, $X \in \mathbb{Z}/N\mathbb{Z}$
- Uses Chebyshev polynomials: $V_n(x + x^{-1}) = x^n + x^{-n}$. Addition rule: $V_{n+m} = V_nV_m - V_{n-m}$, need Lucas chains
- Precompute $V_j(X)$, $1 \le j < d/2$, $j \perp d$ (currently, $d = 210$)
- Compute $V_{id}(X)$, accumulate product of $V_{id}(X) - V_j(X)$ where $id \pm j = q \in \mathbb{P}$, $B_1 < q \le B_2$.
- If $x^q \equiv 1$, $x^{id} \equiv x^j \pmod p$ or $x^{id} \equiv x^{-j} \pmod p$, so $V_{id}(X) - V_j(X) \equiv 0 \pmod p$: gcd finds factor. Allows pairing
- Updating $V_{id}(X) \to V_{(i+1)d}(X) = V_{id}(X) \cdot V_d(X) - V_{(i-1)d}(X)$ takes only 1 multiply. Lucas chain for V_j with 38 mul for 24 j values

ECM stage 2

- Structure very similar to P$\pm$1 stage 2. Curve in Montgomery form, Lucas chains for addition
- Projective coordinates: $(P_x :: P_z) = (Q_x :: Q_z)$ does not imply $P_x = Q_x$, need to cancel z-coordinates
- With stage 1 output P, precompute all required $(id)P$, jP, do batch inversion to normalize

Timings

- Time for finding primes around 2^{27} of 1 word input on 2.126 GHz Core2:

Method	B_1	B_2	Prob.	μs per run	μs per factor
P-1	315	4725	0.168	15.5	92.5
P+1, $x_0 = 2/7$	250	4935	0.172	16.7	88.7
ECM, $\sigma = 10$	150	7875	0.246	39.1	158.8
ECM, $\sigma = 11$	130	6405	0.233	34.4	147.4

- Time for finding primes around 2^{30}:

Method	B_1	B_2	Prob.	μs per run	μs per factor
P-1	400	6405	0.115	18.9	164.1
P+1, $x_0 = 2/7$	350	7245	0.140	22.5	160.4
ECM, $\sigma = 10$	250	12075	0.195	58.8	301.7
ECM, $\sigma = 11$	210	9765	0.179	50.2	281.0

Strategies

- Optimizations for individual methods so far, how to combine P-1, P+1, ECM for maximal effect?
- For each input size (say, a composite of n bits) build a factoring strategy
- For given size n compute expected number of prime factors of m bits
- Example: factor base bound $B = 2^{24}$, large prime bound $L = 2^{30}$, $n = 55$:

m	25	26	27	28	29	30	31
E	0.307	0.305	0.304	0.304	0.304	0.306	0.170

Strategies: Primes

- Factoring algorithms favour primes p in certain residue classes modulo small primes q_i. Obvious for P-1 (prefers $p \equiv 1 \pmod{q_i}$), also for P+1, ECM

- So distinguish prime factors in different residue classes mod small primes, for example here $\pmod{12}$. For GNFS probably uniformly distributed, but not for SNFS

		m						
p	$\pmod{12}$	25	26	27	28	29	30	31
	1	0.0767	0.0762	0.0760	0.0759	0.0761	0.0764	0.0426
	5	0.0767	0.0762	0.0760	0.0759	0.0761	0.0764	0.0426
	7	0.0767	0.0762	0.0760	0.0759	0.0761	0.0764	0.0426
	11	0.0767	0.0762	0.0760	0.0759	0.0761	0.0764	0.0426

Strategies: Probabilities

- For a given method (= an algorithm with particular x_0 / σ, B_1, B_2), compute probability of success for factor sizes, residue class. E.g. P-1 with $B_1 = 400$, $B_2 = 6405$:

		m						
p	$\pmod{12}$	25	26	27	28	29	30	31
	1	0.423	0.364	0.312	0.266	0.225	0.189	0.156
	5	0.291	0.247	0.208	0.174	0.143	0.117	0.094
	7	0.308	0.263	0.222	0.186	0.154	0.126	0.102
	11	0.205	0.171	0.140	0.114	0.091	0.073	0.058

- For $m = 25$, $P + 1$, $x_0 = 2/7$: $0.422, 0.309, 0.309, 0.421$
- For $m = 25$, ECM, $\sigma = 10$: $0.515, 0.392, 0.492, 0.456$
- For $m = 25$, ECM, $\sigma = 11$: $0.515, 0.477, 0.496, 0.455$

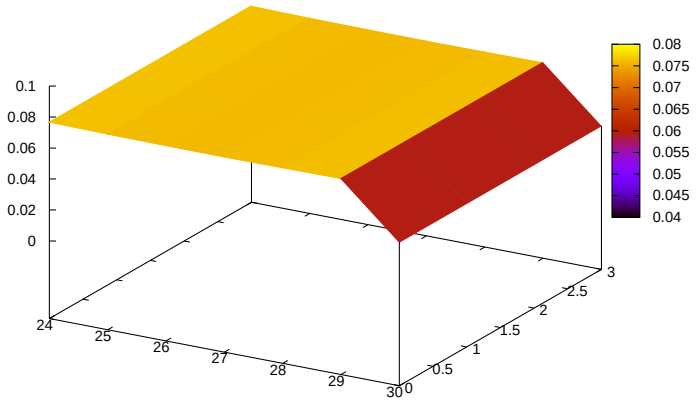
Strategies: Updated primes

- Each factoring attempt changes distribution of prime divisors in remaining composite: Bayes' theorem

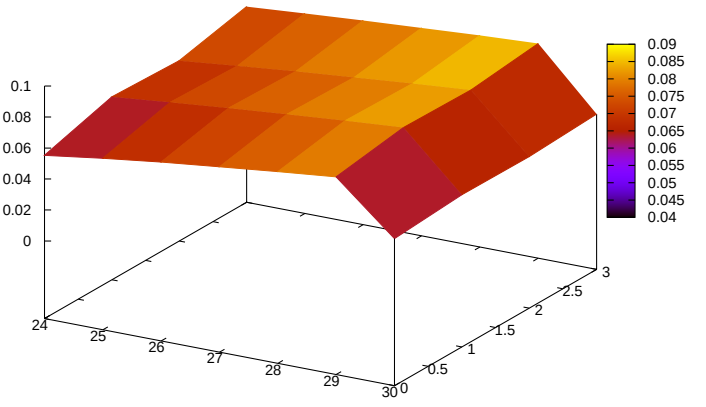
- E.g. after unsucessful P-1 attempt $B_1 = 400$, $B_2 = 6405$, expected numbers of prime factors in each size/residue class become:

		m						
p	$\pmod{12}$	25	26	27	28	29	30	31
	1	0.0553	0.0606	0.0654	0.0696	0.0737	0.0775	0.0449
	5	0.0680	0.0717	0.0752	0.0784	0.0815	0.0843	0.0482
	7	0.0663	0.0702	0.0739	0.0772	0.0805	0.0835	0.0478
	11	0.0762	0.0790	0.0817	0.0841	0.0865	0.0885	0.0502

Before P-1



After P-1



Strategies: Decision

- Given the updated table of expected number of prime divisors, decide what to do next
- If factor was found and cofactor is prime $< L$: success
- If a factor was found, cofactor is prime $> L$ or not possibly smooth: abort
- If cofactor is composite, probability that it is L-smooth too small: abort
- If we continue factoring, choose next method according to updated table

Strategies: Tree

- Strategies are precomputed and stored as tree: nodes are factoring methods or success or abort
- Nodes with factoring method: for each possible cofactor size, edge to next node
- Expected number of factors, probabilities computed only while building strategy tree, actual factoring simply traverses tree
- In NFS, we have two composites that both must be smooth. Nodes specifies which one to try, to minimize time to reach conclusion (factored or not smooth)

Current implementation

- Making good strategies currently in progress. Siever uses dumb strategy right now (a little P-1, a little P+1, then ECM). If $N > L^2$, stop after two ECM curves

- Work in progress, but already benefit from using 3 large primes

- For RSA155, sieving special-q in $[2^{24}, 2^{24} + 1000]$ with $B_r = B_a = 2^{24}$, $L_r = L_a = 2^{30}$:
  Using 2 large primes on both sides $\lambda_r = 2.2, \lambda_a = 2.6$: 640 relations in 94.6 seconds (0.148 s/rel)
  Using 3 large primes on alg. side $\lambda_r = 2.1, \lambda_a = 3.2$: 940 relations in 111.1 seconds (0.118 s/rel), 25% faster