zn_poly: a library for polynomial arithmetic

David Harvey, New York University

January 8, 2009

What is zn_poly?

- `http://cims.nyu.edu/~harvey/zn_poly`
- A free library, written mostly in C, for arithmetic on dense polynomials in $(\mathbf{Z}/m\mathbf{Z})[x]$, where $m \leq$ `LONG_MAX`, e.g. $m < 2^{63}$ on a 64-bit machine
- License: currently GPL; will soon switch to BSD-style
- Under development for about a year
- Portability: included in Sage version ≥ 2.11 (March 2008)
- Depends on GMP
- Threadsafe (no global variables, static variables, ...)

Performance

Following slides compare polynomial multiplication speed for:

- NTL 5.4.2, Mar 2008 (Victor Shoup's free C++ number theory library, `zz_pX` class)
- Magma 2.15-1, Nov 2008 (closed-source computer algebra system with substantial number-theoretic functionality)
- zn_poly 0.9, Oct 2008
- zn_poly 0.10, pre-pre-pre-alpha version

All using GMP 4.2.x with Pierrick Gaudry's AMD64 assembly patches, on a 2.6 GHz Opteron.

x-axis is $\log_2 n$ where $n =$ polynomial degree, y-axis is

$$\log_2 \left(\frac{\text{time in seconds}}{n \log n} \right).$$

Performance

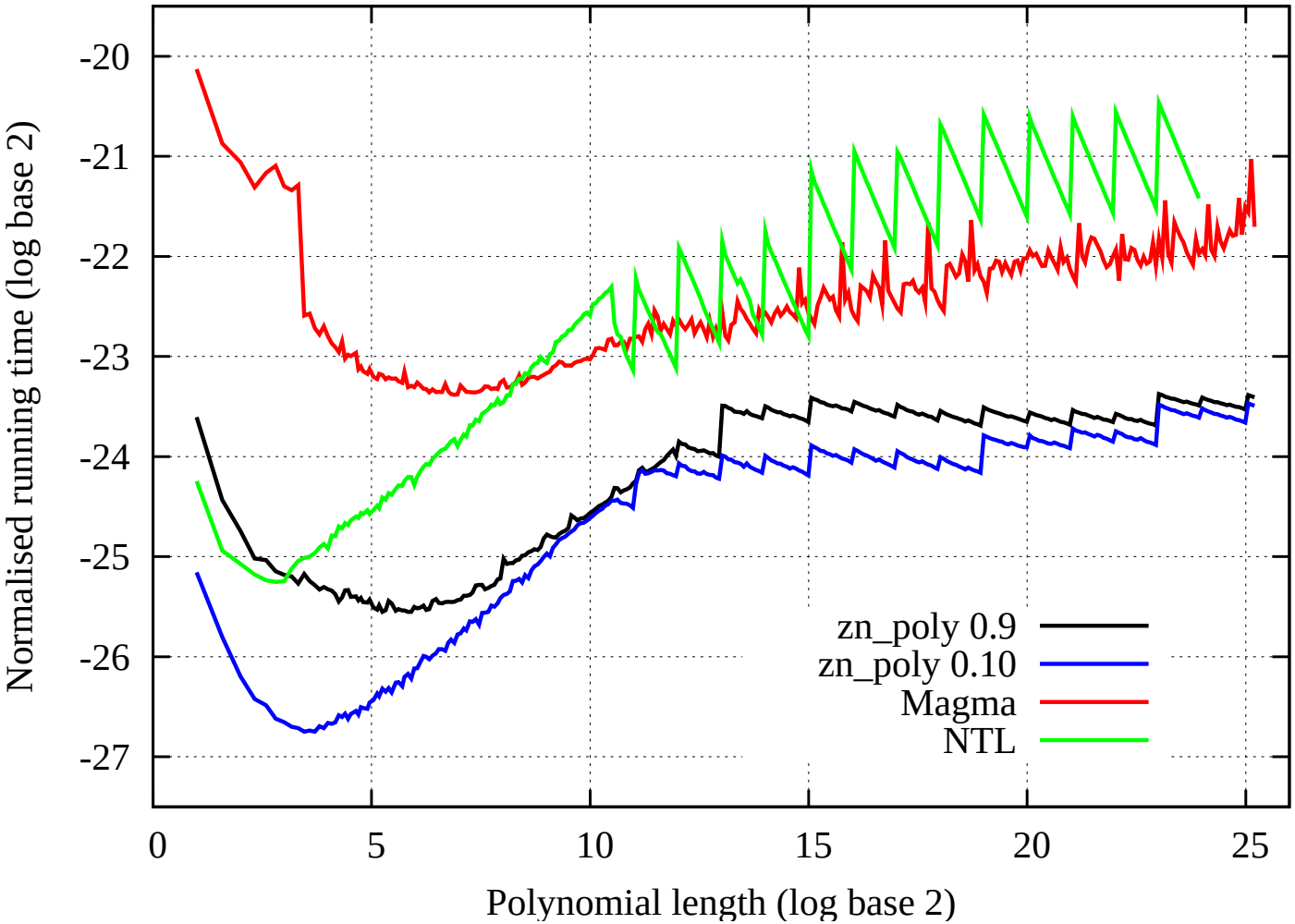

Figure: Multiplication of polynomials modulo $m = \lceil 2^{47.5} \rceil$

Performance

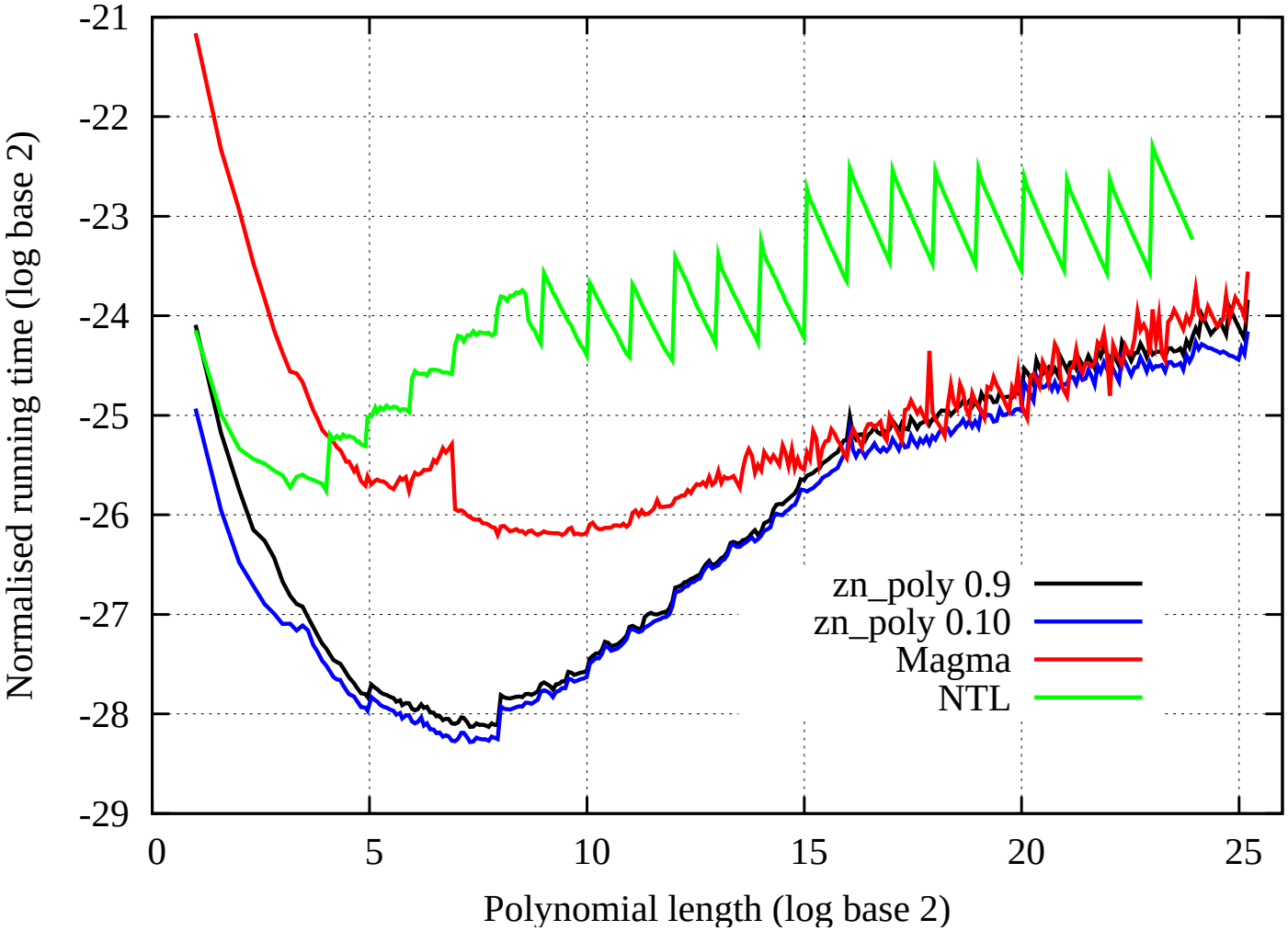

Figure: Multiplication of polynomials modulo $m = 13$

New algorithms

Faster polynomial multiplication via multipoint Kronecker substitution (H., 2007, submitted):

- Ordinary Kronecker substitution reduces polynomial multiplication to integer multiplication: if $f, g \in \mathbf{Z}[x]$ we have $(fg)(2^b) = f(2^b)g(2^b)$; for large enough b this determines fg
- Multipoint Kronecker substitution reduces to several smaller integer multiplications by evaluating at a subset of $\{2^b, -2^b, 2^{-b}, -2^{-b}\}$ (for smaller value of b).

New algorithms

A cache-friendly truncated FFT (H., 2008, submitted):

- Typically, FFT-based methods for multiplying polynomials suffer from jumps in running time when degree crosses a power-of-two boundary
- Van der Hoeven introduced a truncated FFT and inverse truncated FFT (2005) to remove these jumps
- However, his algorithms use divide-and-conquer FFT strategy; not too cache-friendly
- New algorithm combines truncated FFT/IFFT with Bailey's cache-friendly FFT to get the best of both worlds

Applications

Kedlaya's algorithm in larger characteristic, Int. Math. Res. Notices (H., 2007).

A new algorithm for computing the zeta function of a hyperelliptic curve over a finite field.

Record computation using zn_poly: the Jacobian of the curve

$$y^2 = x^7 + 6931471805599453x^6 + \cdots + 1234567890123456 7$$

over GF(p) for $p = 2^{55} - 55$ has

$$2^7 \cdot 3653754110361755444403412585998516510514110342 41$$

points with coordinates in GF(p).

Took 30 hours and used 90 GB RAM.

Applications

Irregular indices, Vandiver's conjecture, and cyclotomic invariants to 163 million (in preparation, joint work with Joe Buhler).

Computed irregular indices and verified Vandiver's conjecture for all primes $p < 163{,}000{,}000$, improving the previous bound of 12,000,000 (Buhler et al, 2001).

Computation time was 21 CPU years at TACC (Texas Advanced Computation Center).

Essentially all the time spent multiplying polynomials over $\mathbf{Z}/p\mathbf{Z}$ of degree $\sim p/2$.

Applications

Other folks who have used zn_poly:

Computing L-series of hyperelliptic curves, Andrew Sutherland and Kiran Kedlaya (MIT), ANTS-VIII, 2008.

Computing Hilbert class polynomials with the Chinese Remainder Theorem (in preparation), Andrew Sutherland.

The FLINT library ("Fast Library for Number Theory", William Hart, Warwick) uses zn_poly for arithmetic in $(\mathbf{Z}/m\mathbf{Z})[x]$.

Example code

To multiply $p(x) = x^3 + 7x$ by $q(x) = x^2 + 4$ by modulo 13:

```
#include "zn_poly.h"

...

zn_mod_t mod;                 // modulus object
zn_mod_init (mod, 13);        // initialise modulus object

unsigned long p[4] = {0, 7, 0, 1};
unsigned long q[3] = {4, 0, 1};
unsigned long r[6];

zn_array_mul (r, p, 4, q, 3, mod);    // multiply raw arrays

zn_mod_clear (mod);
```

The zn_array_* functions operate directly on arrays; they do not perform memory management.

(NTL users: notice that the modulus is passed as a parameter! No global modulus!)

What can zn_poly do today?

zn_array_* layer currently supports:

- Polynomial multiplication
- Polynomial middle product
- Series inversion

Multiplication uses a combination of direct classical/Karatsuba, Kronecker substitution, Schönhage–Nussbaumer FFT. Thresholds tuned automatically, depend on modulus bitsize.

Performance is best for odd moduli.

Roadmap

Version 1.0:

- Expand zn_array_* layer to include division and GCD/XGCD
- Documentation!

Version 2.0:

- zn_poly_t wrapper type with automatic memory management, representation-independent API
- Plug into Sage as back-end for arithmetic in $\mathbf{Z}/m\mathbf{Z}[x]$

Roadmap

Version ≥ 3:

- Improved performance for even moduli, perhaps even $m = 2$
- Packed array format, so e.g. the modulus $m = 3$ doesn't need to use 64 bits per coefficient
- Uniform interface for preconditioned arithmetic; e.g. multiply f by fixed g, multiply f by g modulo fixed h
- Product trees, multipoint evaluation/interpolation, power series composition, modular composition...

Thank you!