Parallel Processing in Algebraic Number Theory

Bill Hart

February 1, 2007

Introduction to FLINT

Fast Library for Number Theory

FLINT: Fast Library for Number Theory

- Jointly Maintained by David Harvey (Harvard) and Bill Hart (Warwick)

FLINT Design Philosophy

- Faster than all available alternatives.

FLINT Design Philosophy

- Faster than all available alternatives.
- Asymptotically Fast Algorithms.

FLINT Design Philosophy

- Faster than all available alternatives.
- Asymptotically Fast Algorithms.
- Library written in C.

FLINT Design Philosophy

- Faster than all available alternatives.
- Asymptotically Fast Algorithms.
- Library written in C.
- Based on GMP.

FLINT Design Philosophy

- Faster than all available alternatives.
- Asymptotically Fast Algorithms.
- Library written in C.
- Based on GMP.
- Extensively Tested.

FLINT Design Philosophy

- Faster than all available alternatives.
- Asymptotically Fast Algorithms.
- Library written in C.
- Based on GMP.
- Extensively Tested.
- Extensively Profiled.

FLINT Design Philosophy

- Faster than all available alternatives.
- Asymptotically Fast Algorithms.
- Library written in C.
- Based on GMP.
- Extensively Tested.
- Extensively Profiled.
- Support for Parallel Processing.

What does FLINT currently do?

- All GMP integer functions (mpz_add → Z_add).

What does FLINT currently do?

- All GMP integer functions (mpz_add $\rightarrow$ Z_add).
- Additional functions for $\mathbb{Z}$ and modulo arithmetic.

What does FLINT currently do?

- All GMP integer functions (mpz_add $\rightarrow$ Z_add).
- Additional functions for $\mathbb{Z}$ and modulo arithmetic.
- Integer Factorisation (Multiple Polynomial Quadratic Sieve).

What does FLINT currently do?

- All GMP integer functions (mpz_add $\rightarrow$ Z_add).
- Additional functions for $\mathbb{Z}$ and modulo arithmetic.
- Integer Factorisation (Multiple Polynomial Quadratic Sieve).
- Some polynomial arithmetic, including asymptotically fast polynomial multiplication.

What does FLINT currently do?

- All GMP integer functions (mpz_add → Z_add).
- Additional functions for $\mathbb{Z}$ and modulo arithmetic.
- Integer Factorisation (Multiple Polynomial Quadratic Sieve).
- Some polynomial arithmetic, including asymptotically fast polynomial multiplication.
- Approximately 21,000 lines of C code so far (including profiling and test code).

What FLINT will eventually do

- Z - Integer Arithmetic.

What FLINT will eventually do

- Z - Integer Arithmetic.
- Zmod - Arithmetic in $\mathbb{Z}/n\mathbb{Z}$.

What FLINT will eventually do

- Z - Integer Arithmetic.
- Zmod - Arithmetic in $\mathbb{Z}/n\mathbb{Z}$.
- Zpoly - Polynomials over $\mathbb{Z}$.

What FLINT will eventually do

- Z - Integer Arithmetic.
- Zmod - Arithmetic in $\mathbb{Z}/n\mathbb{Z}$.
- Zpoly - Polynomials over $\mathbb{Z}$.
- Zvec - Vectors over $\mathbb{Z}$.

What FLINT will eventually do

- Z - Integer Arithmetic.
- Zmod - Arithmetic in $\mathbb{Z}/n\mathbb{Z}$.
- Zpoly - Polynomials over $\mathbb{Z}$.
- Zvec - Vectors over $\mathbb{Z}$.
- Zmat - Linear algebra over $\mathbb{Z}$.

What FLINT will eventually do

- Z - Integer Arithmetic.
- Zmod - Arithmetic in $\mathbb{Z}/n\mathbb{Z}$.
- Zpoly - Polynomials over $\mathbb{Z}$.
- Zvec - Vectors over $\mathbb{Z}$.
- Zmat - Linear algebra over $\mathbb{Z}$.
- Z_p - p-adics.

What FLINT will eventually do

- Z - Integer Arithmetic.
- Zmod - Arithmetic in $\mathbb{Z}/n\mathbb{Z}$.
- Zpoly - Polynomials over $\mathbb{Z}$.
- Zvec - Vectors over $\mathbb{Z}$.
- Zmat - Linear algebra over $\mathbb{Z}$.
- Z_p - p-adics.
- GF2 - Sparse and dense matrices over GF2.

What FLINT will eventually do

- Z - Integer Arithmetic.
- Zmod - Arithmetic in $\mathbb{Z}/n\mathbb{Z}$.
- Zpoly - Polynomials over $\mathbb{Z}$.
- Zvec - Vectors over $\mathbb{Z}$.
- Zmat - Linear algebra over $\mathbb{Z}$.
- Z_p - p-adics.
- GF2 - Sparse and dense matrices over GF2.
- QNF - Quadratic number fields.

What FLINT will eventually do

- Z - Integer Arithmetic.
- Zmod - Arithmetic in $\mathbb{Z}/n\mathbb{Z}$.
- Zpoly - Polynomials over $\mathbb{Z}$.
- Zvec - Vectors over $\mathbb{Z}$.
- Zmat - Linear algebra over $\mathbb{Z}$.
- Z_p - p-adics.
- GF2 - Sparse and dense matrices over GF2.
- QNF - Quadratic number fields.
- NF - General number fields.

What FLINT will eventually do

- Z - Integer Arithmetic.
- Zmod - Arithmetic in $\mathbb{Z}/n\mathbb{Z}$.
- Zpoly - Polynomials over $\mathbb{Z}$.
- Zvec - Vectors over $\mathbb{Z}$.
- Zmat - Linear algebra over $\mathbb{Z}$.
- Z_p - p-adics.
- GF2 - Sparse and dense matrices over GF2.
- QNF - Quadratic number fields.
- NF - General number fields.
- ?? - Whatever people contribute.

Additional Functions Available in FLINT

- Exponentiation.

Additional Functions Available in FLINT

- Exponentiation.
- Modular multiplication, modular inversion, modular square root (mod p or mod p^k), CRT, modular exponentiation.

Additional Functions Available in FLINT

- Exponentiation.
- Modular multiplication, modular inversion, modular square root (mod p or mod p^k), CRT, modular exponentiation.
- Next prime, random prime, extended GCD, GCD.

Additional Functions Available in FLINT

- Exponentiation.
- Modular multiplication, modular inversion, modular square root (mod p or mod p^k), CRT, modular exponentiation.
- Next prime, random prime, extended GCD, GCD.
- Integer multiplication (faster than GMP 4.2.1 with Pierrick Gaudry's AMD64 patches for more than 164000 bit operands).

Additional Functions Available in FLINT

- Exponentiation.
- Modular multiplication, modular inversion, modular square root (mod p or mod p^k), CRT, modular exponentiation.
- Next prime, random prime, extended GCD, GCD.
- Integer multiplication (faster than GMP 4.2.1 with Pierrick Gaudry's AMD64 patches for more than 164000 bit operands).
- Block Lanczos code (Jason Papadopoulous).

Additional Functions Available in FLINT

- Exponentiation.
- Modular multiplication, modular inversion, modular square root (mod p or mod p^k), CRT, modular exponentiation.
- Next prime, random prime, extended GCD, GCD.
- Integer multiplication (faster than GMP 4.2.1 with Pierrick Gaudry's AMD64 patches for more than 164000 bit operands).
- Block Lanczos code (Jason Papadopoulous).
- Polynomial root finding code (Jason P. - not yet integrated).

Additional Functions Available in FLINT

- Exponentiation.
- Modular multiplication, modular inversion, modular square root (mod p or mod p^k), CRT, modular exponentiation.
- Next prime, random prime, extended GCD, GCD.
- Integer multiplication (faster than GMP 4.2.1 with Pierrick Gaudry's AMD64 patches for more than 164000 bit operands).
- Block Lanczos code (Jason Papadopoulous).
- Polynomial root finding code (Jason P. - not yet integrated).
- SQUFOF factoring algorithm (Jason P. - not yet integrated).

Additional Functions Available in FLINT

- Exponentiation.
- Modular multiplication, modular inversion, modular square root (mod p or mod p^k), CRT, modular exponentiation.
- Next prime, random prime, extended GCD, GCD.
- Integer multiplication (faster than GMP 4.2.1 with Pierrick Gaudry's AMD64 patches for more than 164000 bit operands).
- Block Lanczos code (Jason Papadopoulous).
- Polynomial root finding code (Jason P. - not yet integrated).
- SQUFOF factoring algorithm (Jason P. - not yet integrated).
- Self initialising multiple polynomial quadratic sieve (for integer factorization).

Additional Functions Available in FLINT

- Exponentiation.
- Modular multiplication, modular inversion, modular square root (mod p or mod p^k), CRT, modular exponentiation.
- Next prime, random prime, extended GCD, GCD.
- Integer multiplication (faster than GMP 4.2.1 with Pierrick Gaudry's AMD64 patches for more than 164000 bit operands).
- Block Lanczos code (Jason Papadopoulous).
- Polynomial root finding code (Jason P. - not yet integrated).
- SQUFOF factoring algorithm (Jason P. - not yet integrated).
- Self initialising multiple polynomial quadratic sieve (for integer factorization).
- Memory management for single mpz_t's and arrays of mpz_t's, arrays of limbs.

Polynomial Arithmetic Available so far

- Allocate, deallocate, copy, clear.

Polynomial Arithmetic Available so far

- Allocate, deallocate, copy, clear.
- Maximum coefficient size, whether coefficients are signed or unsigned, maximum length.

Polynomial Arithmetic Available so far

- Allocate, deallocate, copy, clear.
- Maximum coefficient size, whether coefficients are signed or unsigned, maximum length.
- Add, subtract, multiply by scalar.

Polynomial Arithmetic Available so far

- Allocate, deallocate, copy, clear.
- Maximum coefficient size, whether coefficients are signed or unsigned, maximum length.
- Add, subtract, multiply by scalar.
- Truncate, rotate.

Polynomial Arithmetic Available so far

- Allocate, deallocate, copy, clear.
- Maximum coefficient size, whether coefficients are signed or unsigned, maximum length.
- Add, subtract, multiply by scalar.
- Truncate, rotate.
- Polynomial multiplication (including Karatsuba, RadixMul, Schoenhage-Strassen, Kronecker-Strassen).

Polynomial Arithmetic Available so far

- Allocate, deallocate, copy, clear.
- Maximum coefficient size, whether coefficients are signed or unsigned, maximum length.
- Add, subtract, multiply by scalar.
- Truncate, rotate.
- Polynomial multiplication (including Karatsuba, RadixMul, Schoenhage-Strassen, Kronecker-Strassen).
- Many test and profiling functions.

Why do we need a new Library?

- What about Pari, NTL, LiDIA, others?

Why do we need a new Library?

- What about Pari, NTL, LiDIA, others?
- What about MAGMA, MAPLE, Mathematica, etc?

Why do we need a new Library?

- What about Pari, NTL, LiDIA, others?
- What about MAGMA, MAPLE, Mathematica, etc?
- SAGE seems to be doing just fine building in functionality from NTL and Pari and others.

Sieve timing comparisons

Digits	Msieve	FLINT	Pari
C41	0.33s	0.24s	0.34s
C51	1.4s	1.4s	3.78s
C61	9s	15.6s	61.3s
C71	90s	187s	392s
C81	820s	2160s	7985s
C86	4200s	7380s	Umm yeah

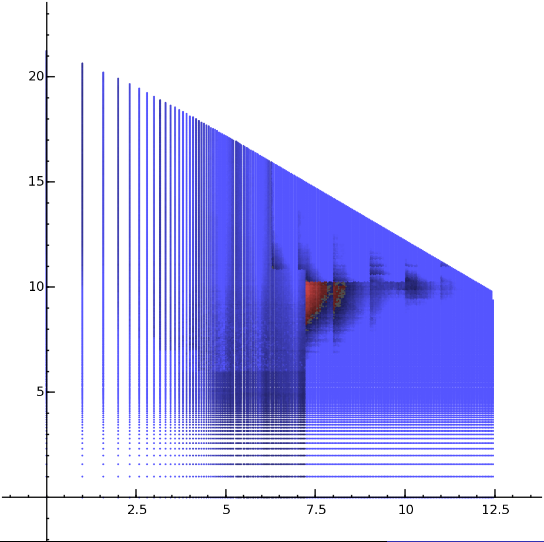
Timings for a 1.8GHz Opteron (sage.math)

Sieve timing comparisons

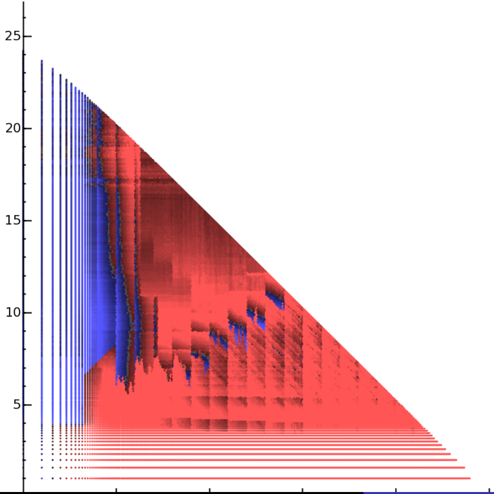
Digits	Msieve	FLINT	Pari
C41	0.44s	0.40s	1.1s
C51	1.97s	1.82s	5.5s
C61	13s	18s	90s
C71	133s	187s	690s
C76	568s	898	2970s
C81	1045s	2320s	7920s
C86	5880s	8580s	Ahem

Timings for an Athlon XP 2000+ (laptop)

Polynomial Multiplication: NTL vs Pari (Pari = Red)



Polynomial Multiplication: MAGMA vs NTL (MAGMA = Red)



Algorithms for Polynomial Multiplication

- Radix Multiplication (used by NTL - old algorithm)

Algorithms for Polynomial Multiplication

- Radix Multiplication (used by NTL - old algorithm)
- Schoenhage-Strassen (based on FFT's)

Algorithms for Polynomial Multiplication

- Radix Multiplication (used by NTL - old algorithm)
- Schoenhage-Strassen (based on FFT's)
- Kronecker-Schoenhage (combine into large integers and multiply)

Algorithms for Polynomial Multiplication

- Radix Multiplication (used by NTL - old algorithm)
- Schoenhage-Strassen (based on FFT's)
- Kronecker-Schoenhage (combine into large integers and multiply)
- Karatsuba

Algorithms for Polynomial Multiplication

- Radix Multiplication (used by NTL - old algorithm)
- Schoenhage-Strassen (based on FFT's)
- Kronecker-Schoenhage (combine into large integers and multiply)
- Karatsuba
- Toom

Karatsuba Method

$$(a_1 + a_2 x^n)(b_1 + b_2 x^n) = a_1 b_1 + a_2 b_2 x^{2n} + (a_1 + a_2)(b_1 + b_2)x^n - a_1 b_1 x^n - a_2 b_2 x^n$$

Schoenhage-Strassen Method

- A polynomial of degree n is completely determined by its values at $n+1$ distinct points.

Schoenhage-Strassen Method

- A polynomial of degree n is completely determined by its values at $n+1$ distinct points.
- $g(x) = f_1(x) * f_2(x)$ is determined by its value at $2n$ points, if f_1, f_2 have length n.

Schoenhage-Strassen Method

- A polynomial of degree n is completely determined by its values at $n+1$ distinct points.
- $g(x) = f_1(x) * f_2(x)$ is determined by its value at $2n$ points, if f_1, f_2 have length n.
- Discrete Fourier Transform chooses $2n$-th roots of unity as the points to evaluate at.

Schoenhage-Strassen Method

- A polynomial of degree n is completely determined by its values at $n+1$ distinct points.
- $g(x) = f_1(x) * f_2(x)$ is determined by its value at $2n$ points, if f_1, f_2 have length n.
- Discrete Fourier Transform chooses $2n$-th roots of unity as the points to evaluate at.
- Compute DFT of coefficients of f_1, compute DFT of coefficients of f_2, multiply the $2n$ values, perform an inverse transform.

Schoenhage-Strassen Method

- A polynomial of degree n is completely determined by its values at $n+1$ distinct points.
- $g(x) = f_1(x) * f_2(x)$ is determined by its value at $2n$ points, if f_1, f_2 have length n.
- Discrete Fourier Transform chooses $2n$-th roots of unity as the points to evaluate at.
- Compute DFT of coefficients of f_1, compute DFT of coefficients of f_2, multiply the $2n$ values, perform an inverse transform.
- FFT is a method for computing the DFT quickly.

Schoenhage-Strassen Method

- A polynomial of degree n is completely determined by its values at $n+1$ distinct points.
- $g(x) = f_1(x) * f_2(x)$ is determined by its value at $2n$ points, if f_1, f_2 have length n.
- Discrete Fourier Transform chooses $2n$-th roots of unity as the points to evaluate at.
- Compute DFT of coefficients of f_1, compute DFT of coefficients of f_2, multiply the $2n$ values, perform an inverse transform.
- FFT is a method for computing the DFT quickly.
- Schoenhage-Strassen technique works in the ring $\mathbb{Z}/(2^n+1)\mathbb{Z}$, for which 2 is a $2n$-th root of unity.

Schoenhage-Strassen Method

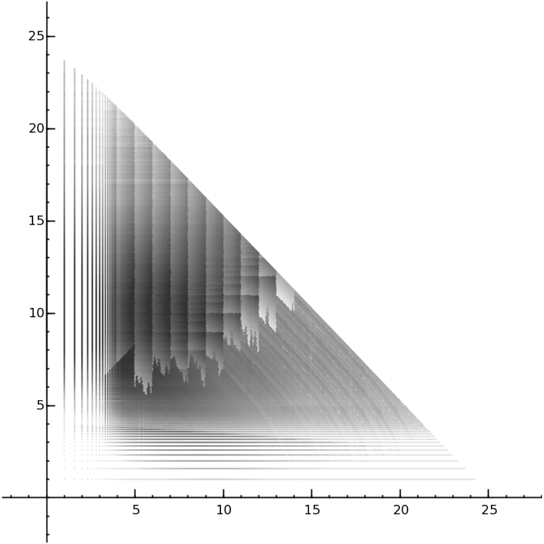
- A polynomial of degree n is completely determined by its values at $n+1$ distinct points.
- $g(x) = f_1(x) * f_2(x)$ is determined by its value at $2n$ points, if f_1, f_2 have length n.
- Discrete Fourier Transform chooses $2n$-th roots of unity as the points to evaluate at.
- Compute DFT of coefficients of f_1, compute DFT of coefficients of f_2, multiply the $2n$ values, perform an inverse transform.
- FFT is a method for computing the DFT quickly.
- Schoenhage-Strassen technique works in the ring $\mathbb{Z}/(2^n+1)\mathbb{Z}$, for which 2 is a $2n$-th root of unity.
- Multiplications by roots of unity are now just bitshifts.

FFT(A, m, w):

A = vector length m, w = primitive m-th root of unity

```
if (m==1) return vector (a_0)
else {
  A_even = (a_0, a_2, ..., a_{m-2})
  A_odd  = (a_1, a_3, ..., a_{m-1})
  F_even = FFT(A_even, m/2, w^2)
  F_odd = FFT(A_odd, m/2, w^2)
  F = new vector of length m
  x = 1
  for (j=0; j < m/2; ++j) {
    F[j] = F_even[j] + x*F_odd[j]
    F[j+m/2] = F_even[j] - x*F_odd[j]
    x = x * w
}
return F
```

What does MAGMA use?



What we do

- Variants of Schoenhage-Strassen and Kronecker-Schoenhage.

What we do

- Variants of Schoenhage-Strassen and Kronecker-Schoenhage.
- Trick suggested by David Harvey and Paul Zimmerman for KS.

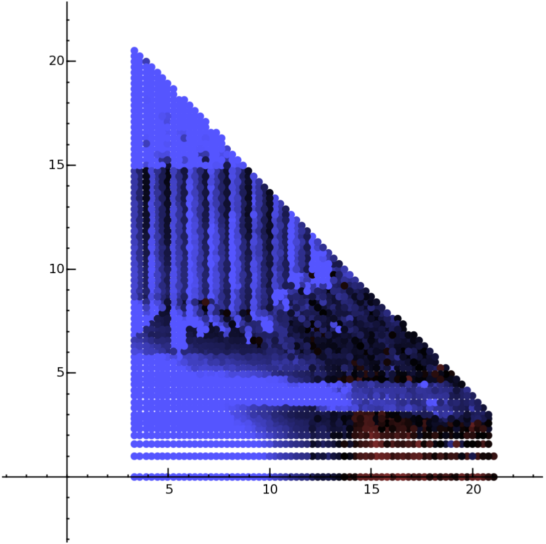
What we do

- Variants of Schoenhage-Strassen and Kronecker-Schoenhage.
- Trick suggested by David Harvey and Paul Zimmerman for KS.
- Bailey's four-step algorithm.

What we do

- Variants of Schoenhage-Strassen and Kronecker-Schoenhage.
- Trick suggested by David Harvey and Paul Zimmerman for KS.
- Bailey's four-step algorithm.
- Truncated FFT (with 2-step) - Joris van der Hoeven.

FLINT vs MAGMA



Parallelisation

- No global or static variables.

Parallelisation

- No global or static variables.
- Memory management (needs to support multiple threads requesting memory).

Parallelisation

- No global or static variables.
- Memory management (needs to support multiple threads requesting memory).
- Posix threads.

Parallelisation

- No global or static variables.
- Memory management (needs to support multiple threads requesting memory).
- Posix threads.
- Very next version of GCC will support OpenMP.

Parallelisation

- No global or static variables.
- Memory management (needs to support multiple threads requesting memory).
- Posix threads.
- Very next version of GCC will support OpenMP.
- Quadratic sieve can use disk based parallelism.

Our hackish attempt at pthreads

- Frustration at the lack of open source mathematics that use pthreads.

Our hackish attempt at pthreads

- Frustration at the lack of open source mathematics that use pthreads.
- Read that 200,000 threads can be started by the kernel, per second.

Our hackish attempt at pthreads

- Frustration at the lack of open source mathematics that use pthreads.
- Read that 200,000 threads can be started by the kernel, per second.
- Threads may take some time to be scheduled (real-time threads).

Some solutions?

- Queue of jobs from which threads can pull tasks.

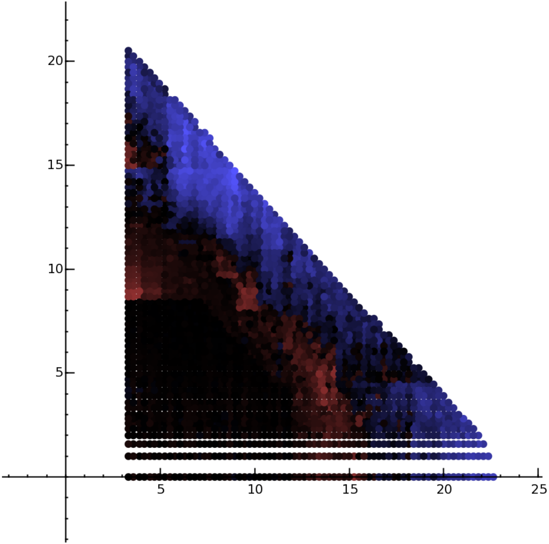
Some solutions?

- Queue of jobs from which threads can pull tasks.
- Threads go to sleep when there is no work and wake up when a condition is met.

Some solutions?

- Queue of jobs from which threads can pull tasks.
- Threads go to sleep when there is no work and wake up when a condition is met.
- For some problems, threads should not be used.

Two Threads versus None



Four Threads versus Two Threads