Note Taker Checklist Form -*MSRI*

Name : Rob Stapleton

E-mail Address/ Phone #: jrstaple@ncsu.edu

Talk Title and Workshop assigned to:
Interactive Parallel Computing In Support of Research in Algebra, Geometry and Number Theory

Lecturer (Full name): Bill Hart

Date & Time of Event: _____

Check List:

(✓) Introduce yourself to the lecturer prior to lecture. Tell them that you will be the note taker, and that you will need to make copies of their own notes, if any.

(✓) Obtain all presentation materials from lecturer (i.e. Power Point files, etc). This can be done either before the lecture is to begin or after the lecture; please make arrangements with the lecturer as to when you can do this.

(✓) Take down all notes from media provided (blackboard, overhead, etc.)

(✓) Gather all other lecture materials (i.e. Handouts, etc.)

(✓) Scan all materials on PDF scanner in 2nd floor lab (assistance can be provided by Computing Staff) – Scan this sheet first, then materials. In the subject heading, enter the name of the speaker and date of their talk.

Please do **NOT** use **pencil** or colored pens other than black when taking notes as the scanner has a difficult time scanning pencil and other colors.

Please fill in the following after the lecture is done:

1. List 6-12 lecture keywords: Parallel, interactive FLINT, number theory, MAGMA, FFT, Pari, SAGE

2. Please summarize the lecture in 5 or less sentences.
Number Theorists need shiny, slick, parallel toys, too. This is where FLINT comes in hand tries to beat the competition.

Once the materials on check list above are gathered, please scan ALL materials and send to the Computing Department. Return this form to Larry Patague, Head of Computing (rm 214)

10:30 a.m.
Parallel Processing in Algebraic Number Theory
Bill Hart

FLINT = Fast Library for Number Theory. Jointly maintained by David Harvey and Bill Hart.
Design Philosophy
- -Want to be faster than all available alternatives (unrealistic and unachievable)
- -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
- -Additional functions for integer and modular arithmetic
  - -Not efficient for small operands. GMP is very efficient for very long integers.
- -Integer factorisation (Multiple Polynomial Quadratic Sieve)
- -Some polynomial arithmetic, including asymptotically fast polynomial multiplication

What will FLINT eventually do?
- -Z-integer arithmetic
- -Zmod - Arithmetic in the intergers modulo n
- -Zpoly - Polynomials over the integers
- -Zvec - vectors over the integers
- -Zmat - Linear algebra over the integers
- -Z_p - p-adics
- -GF2 - Sparse and dense matricies over GF2
- -QNF - Quadratic Number Fields
- -NF - Genreal Number Fields

Additional functions available in FLINT
- exponentiation
- modular multiplication, inversion, square rooot, CRT, exponentiation
- next prime, random prime, extended GCD, GCD
- integer multiplication (faster than GMP 4.2.1)
- Block Lanczos code
- polynomial root finding code
- SQUFOF factoring algorithm
- self initialising multiple polynomial quadratic sieve
- memory management for single mpz_t's and arrays of mpz_t's

Basic polynomial arithmetic is available.

Why a new library? What about Pari, NTL, LiDIA, others? What about Magma, Maple, Mathematica? What about SAGE?

FLINT currently beats Pari in integer factorisation, and is faster (at smaller sizes) than Msieve.

FLINT beats Pari virtually all the time at polynomial multiplication. MAGMA beats NTL at polynomial multiplication similarly

Algorithms for Polynomial Multiplication
- -Radix multiplications (used by NTL, old algorithm, outdated)

-Schoenhage-Strassen (based on FFTs)
-Kronecker-Schoenhage (combine into large integers and multiply) (most efficient algorithm for certain problem sizes)
-Karatsuba
-Toom

All of the talk of polynomial multiplications is only of the univariate case.

Schoenhage-Strassen Method:

A polynomail 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 of f_1 and f_2 have length n

Discrete FFT chooses 2n-th roots of unity as the poitns at which to evaluate.

Compute DFT of coefficients of f_1, compute DFT of coefficients of f_2, multiply the 2n values, perform inverse transformation.

Use FFt.

Schoenhage-Strassen works in the ring integers modulo (2^n+1)

What does MAGMA use? By looking at a graph of the expected times vs actual performance times, it seems that MAGMa uses Schoenhage-Strassen by the jumps of runtime at lengths of powers of 2. Something else seems to be going on when one polynomial is small.

What does FLINT do? Flint uses variants of Schoenhage-Strassen and Kroenecker-Strassen.

Uses a trick suggested by David Harvey and Paul Zimmerman for KS.

Uses Bailey's four-step algorithm.

Truncated FFT (with 2 step) - Joris van der Hoeven

After some work, FLINT now competes with MAGMA.

Parallelisation in FLINT:

No global or static variables

Memory management (needs to support multiple threads requesting memory).

Decided to use Posix threads for now.

There is a frustratoin at the lack of open source mathematics that uses pthreads.
There are (outrageous) claims that 200,000 threads cna 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

-Threads go to sleep when thre is no work and wake up when a condition is met

-Sometimes, they're just not necessary

For parallelising the FFT, the general idea is to break the vectors into smaller chunks and performing FFTs on those chunks.

Feeling much more confident now that FLINT is doable and even parallelisable

SAGE relies on Pari for many things. Would like to do a lot of these things themselves over working with Pari to improve Pari
once you introduce a layer of abstraction into a problem, you need C++

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

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

ieve 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

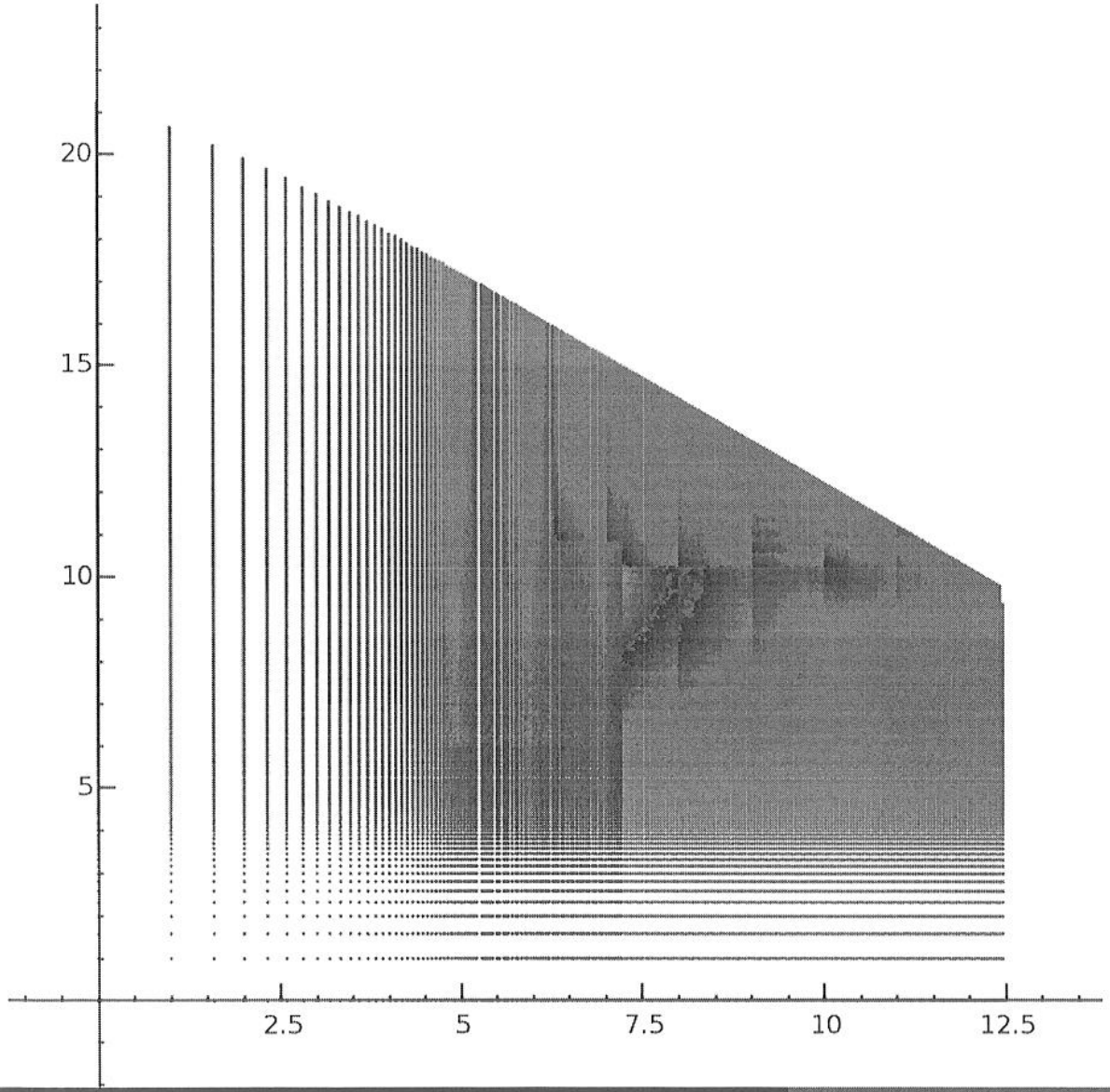
Timings for a 1.8GHz Opteron (sage.math)

Sieve timing comparisons

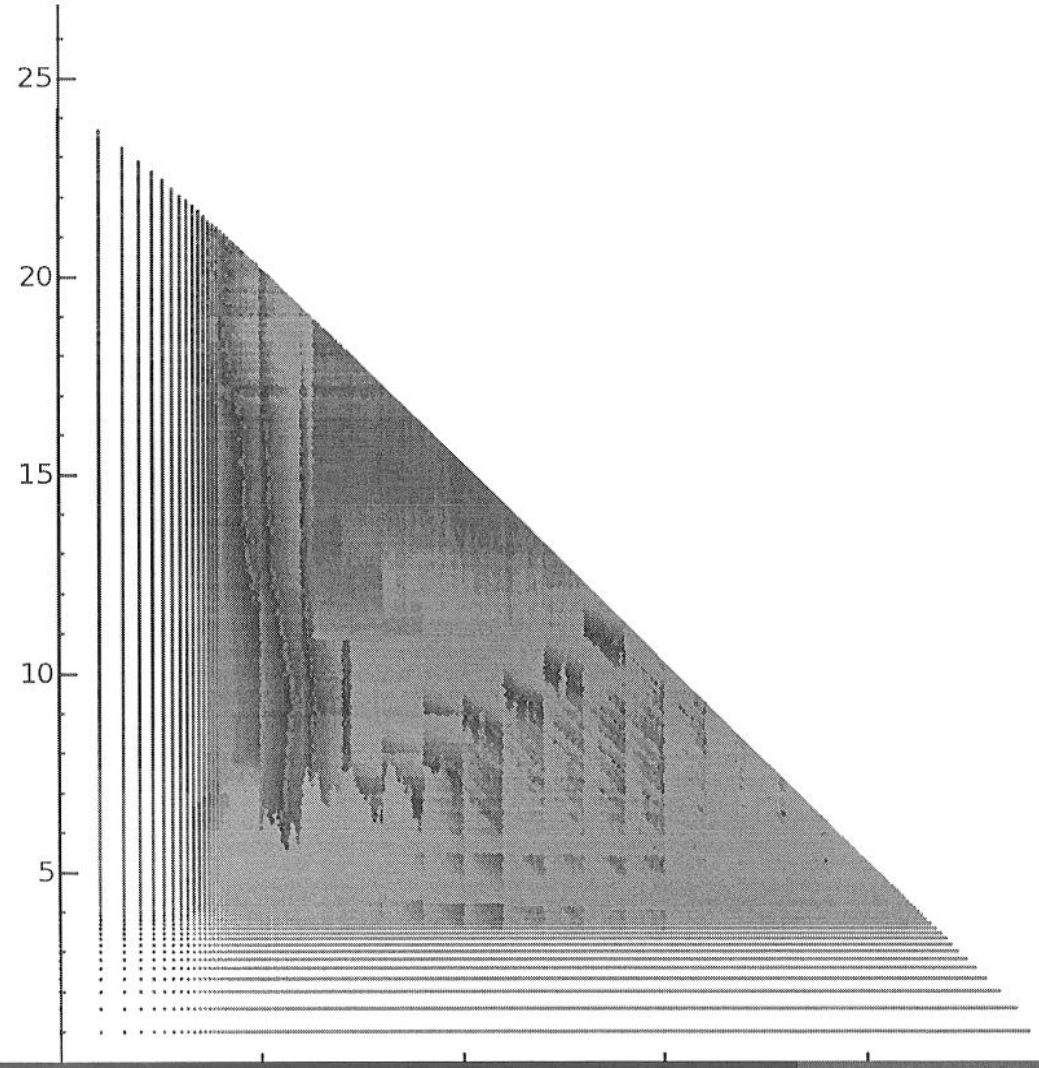
Digits	Msieve	FLINT	Pari
C41	0.44s	0.40s	1.1s
C51	1.97s	1.82s	5.5s
C61	13s	18s	90s
C71	133s	187s	690s
C76	568s	898	2970s
C81	1045s	2320s	7920s
C86	5880s	8580s	Ahem

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)
- 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

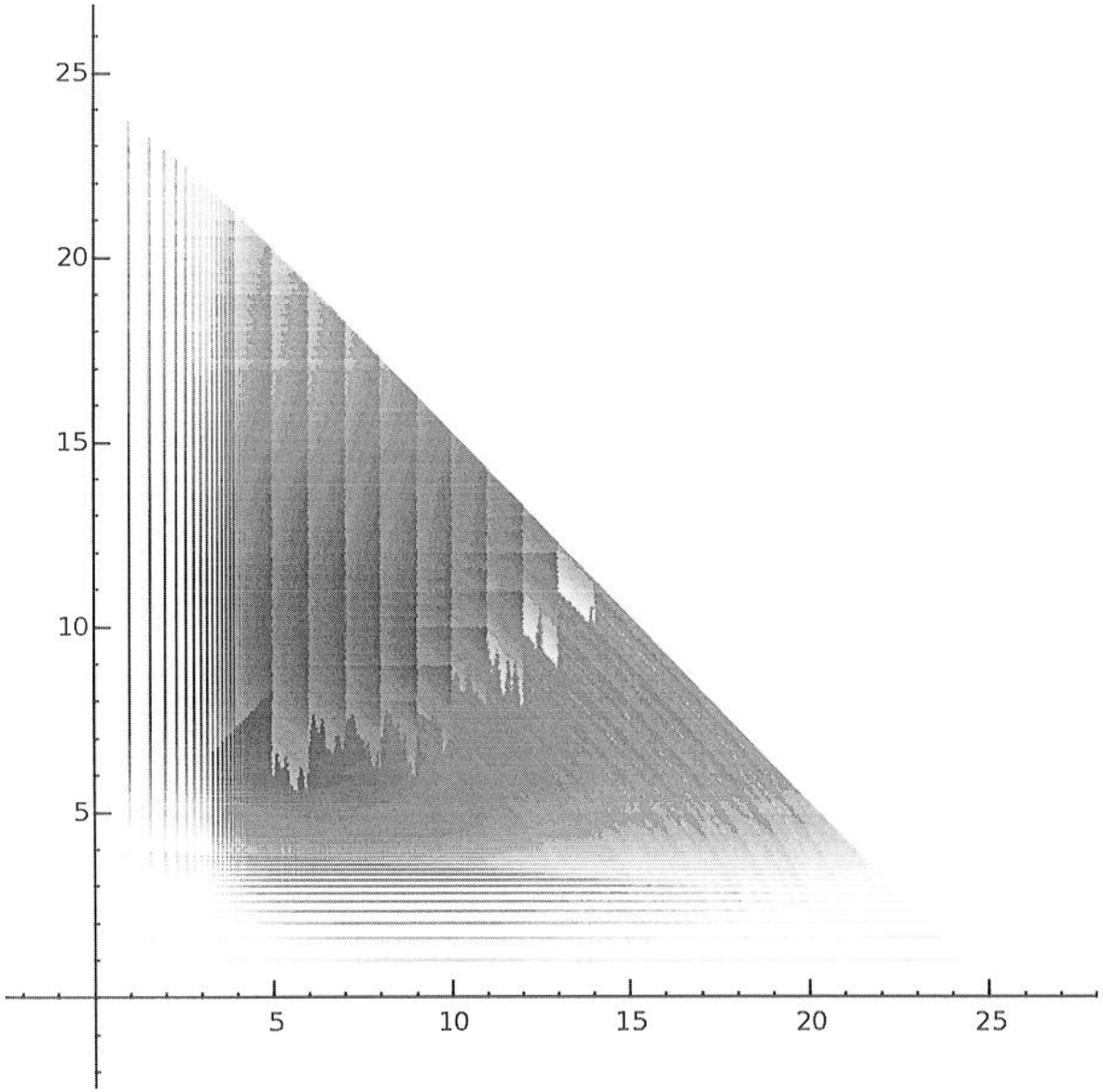
- A polynomial of degree n is completely determined by its values at $n+1$ distinct points.
- $g(x) = f_1(x) * f_2(x)$ is determined by its value at $2n$ points, if f_1, f_2 have length n.
- Discrete Fourier Transform chooses $2n$-th roots of unity as the points to evaluate at.
- Compute DFT of coefficients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

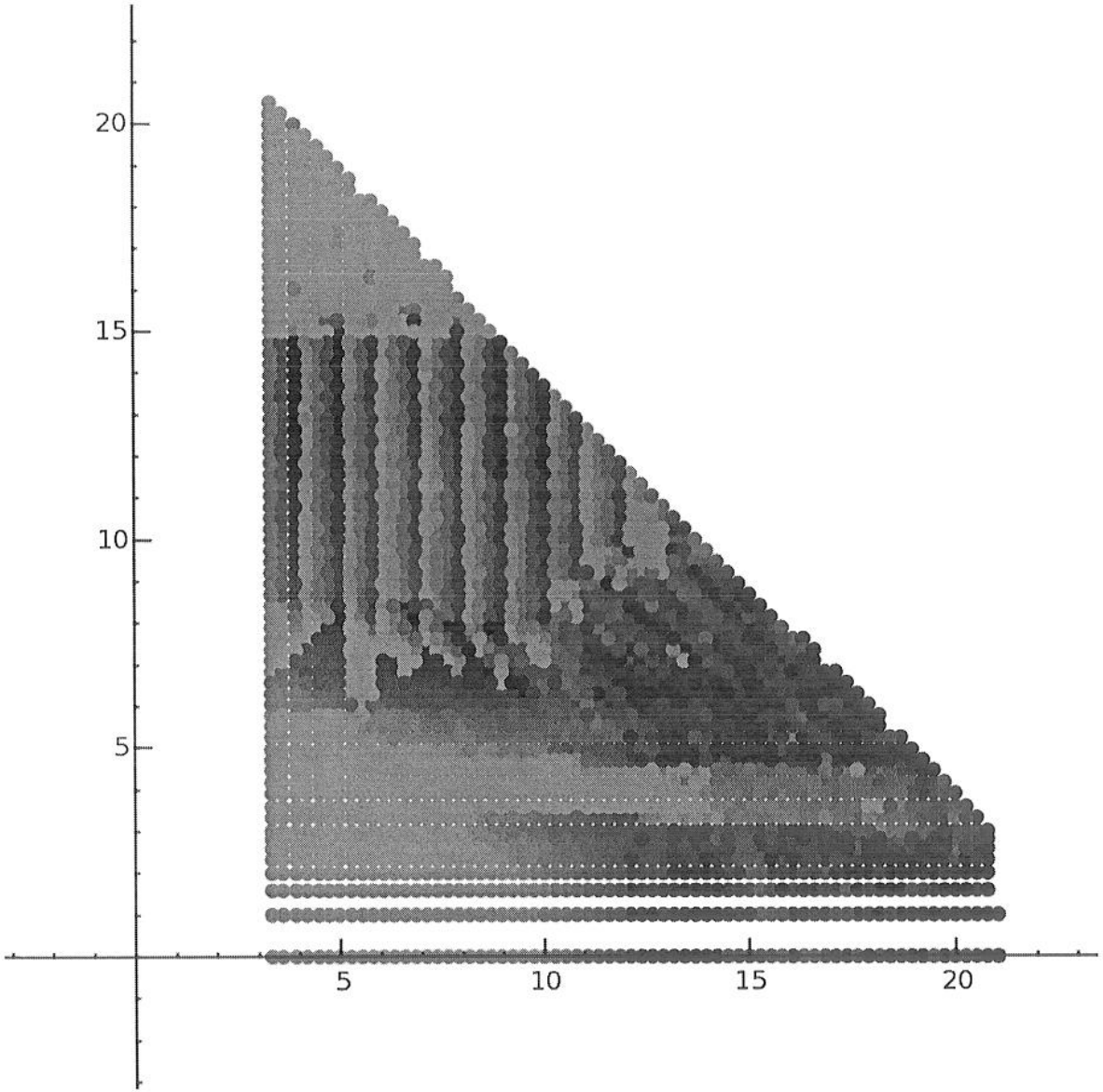
What does MAGMA use?



What we do

- Variants of Schoenhage-Strassen and Kronecker-Schoenhage.
- 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.
- 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.

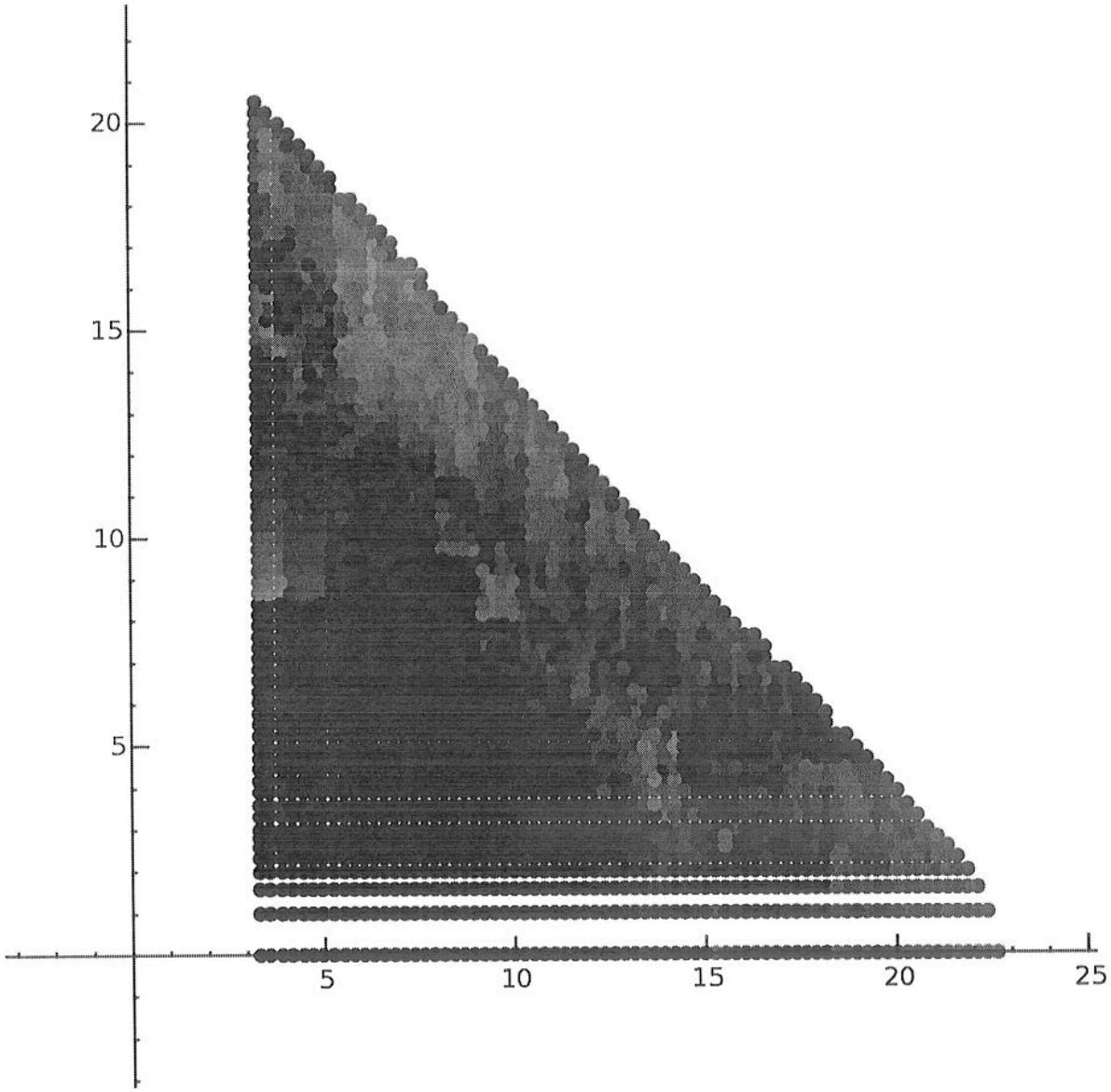
Our hackish attempt at pthreads

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

ome 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