The FFLAS-FFPACK and LinBox libraries

OpenDreamKit Software presentation

Clément Pernet & the LinBox group

September 2, 2015

Exact linear algebra

Matrices can be

Dense: store all coefficients

Sparse: store the non-zero coefficients only

Black-box: no access to the storage, only *apply* to a vector

Exact linear algebra

Matrices can be

Dense: store all coefficients

Sparse: store the non-zero coefficients only

Black-box: no access to the storage, only *apply* to a vector

Coefficient domains:

Word size:
- integers with a priori bounds
- $\mathbb{Z}/p\mathbb{Z}$ for p of ≈ 32 bits

Multi-precision: $\mathbb{Z}/p\mathbb{Z}$ for p of $\approx 100, 200, 1000, 2000, \dots$ bits

Arbitrary precision: $\mathbb{Z}, \mathbb{Q}$

Polynomials: $\mathsf{K}[X]$ for K any of the above

Exact linear algebra

Matrices can be

Dense: store all coefficients

Sparse: store the non-zero coefficients only

Black-box: no access to the storage, only *apply* to a vector

Coefficient domains:

Word size:
- integers with a priori bounds
- $\mathbb{Z}/p\mathbb{Z}$ for p of ≈ 32 bits

Multi-precision: $\mathbb{Z}/p\mathbb{Z}$ for p of $\approx 100, 200, 1000, 2000, \dots$ bits

Arbitrary precision: $\mathbb{Z}, \mathbb{Q}$

Polynomials: K$[X]$ for K any of the above

Several implementations for the same domain: better fits FFT, LinAlg, etc

Exact linear algebra

Matrices can be

Dense: store all coefficients

Sparse: store the non-zero coefficients only

Black-box: no access to the storage, only *apply* to a vector

Coefficient domains:

Word size:
- integers with a priori bounds
- $\mathbb{Z}/p\mathbb{Z}$ for p of ≈ 32 bits

Multi-precision: $\mathbb{Z}/p\mathbb{Z}$ for p of $\approx 100, 200, 1000, 2000, \ldots$ bits

Arbitrary precision: $\mathbb{Z}, \mathbb{Q}$

Polynomials: K$[X]$ for K any of the above

Several implementations for the same domain: better fits FFT, LinAlg, etc

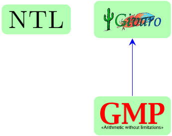
Requires genericity.

Software stack for exact linear algebra

Arithmetic

GMP «Arithmetic without limitations», MPIR: multiprecision integers and rationals

Givaro, NTL: finite fields and polynomials



Software stack for exact linear algebra

Arithmetic

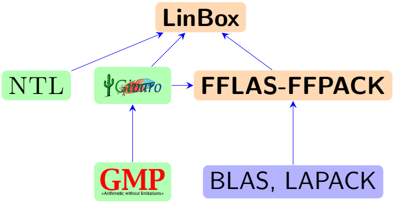
GMP, MPIR: multiprecision integers and rationals

Givaro, NTL: finite fields and polynomials

BLAS: Basic Linear Algebra Subroutines (floating point)

FFLAS-FFPACK: Basic Exact Linear Algebra over $\mathbb{Z}/p\mathbb{Z}$,

LinBox: Linear Algebra over $\mathbb{Z}, \mathbb{Z}/p\mathbb{Z}$ and $K[X]$



Software stack for exact linear algebra

Arithmetic

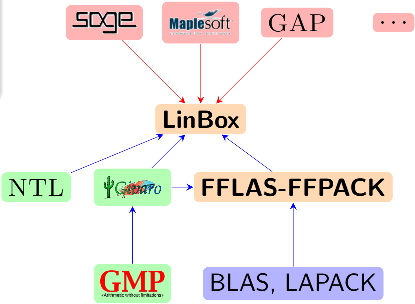
GMP, MPIR: multiprecision integers and rationals

Givaro, NTL: finite fields and polynomials

BLAS: Basic Linear Algebra Subroutines (floating point)

FFLAS-FFPACK: Basic Exact Linear Algebra over $\mathbb{Z}/p\mathbb{Z}$,

LinBox: Linear Algebra over $\mathbb{Z}, \mathbb{Z}/p\mathbb{Z}$ and $\mathrm{K}[X]$



Outline

The LinBox project

- International collaboration: Canada, USA, France
- Strongly generic `C++` code, focus on efficiency
- Free software (LGPL 2.1+)
- ≈ 200 K loc
- `http://linalg.org/`

The LinBox project

- International collaboration: Canada, USA, France
- Strongly generic `C++` code, focus on efficiency
- Free software (LGPL 2.1+)
- ≈ 200 K loc
- `http://linalg.org/`

Milestones

1998	First design: Black box and sparse matrices
2003	Dense linear algebra using BLAS $\rightsquigarrow$ FFLAS-FFPACK
2005	LinBox-1.0
2008	Integration in Sage
2012-..	Parallelization
2014	SIMD & Sparse BLAS in FFLAS-FFPACK (Brice's talk)

Outline

Black box linear algebra

Black box linear algebra

- Matrices viewed as linear operators
- algorithms based on matrix-vector apply only $\rightsquigarrow$ cost $E(n)$

Black box linear algebra

- Matrices viewed as linear operators
- algorithms based on matrix-vector apply only $\rightsquigarrow$ cost $E(n)$

Structured matrices: Fast apply (e.g. $E(n) = O(n \log n)$)

Sparse matrices: Fast apply and no fill-in

Black box linear algebra

- Matrices viewed as linear operators
- algorithms based on matrix-vector apply only $\rightsquigarrow$ cost $E(n)$

Structured matrices: Fast apply (e.g. $E(n) = O(n \log n)$)

Sparse matrices: Fast apply and no fill-in

$\rightsquigarrow$

- Iterative methods
- No access to coefficients, trace, no elimination
- Matrix **multiplication** $\Rightarrow$ Black-box **composition**

Example: blackbox composition

```
template <class Mat1, class Mat2>
class Compose {
  protected:
    Mat1 _A;
    Mat2 _B;
  public:
    Compose(Mat1& A, Mat2& B) : _A(A), _B(B) {}

    template<class InVec, class OutVec>
    OutVec& apply (const InVec& x) {
      return _A.apply(_B.apply(x));
    }
};
```

Black box linear algebra

Matrix-Vector Product: building block,
$\rightsquigarrow$ costs $E(n)$

Minimal polynomial: [Wiedemann 86]
$\rightsquigarrow$ iterative Krylov/Lanczos methods
$\rightsquigarrow O(nE(n) + n^2)$

Black box linear algebra

Matrix-Vector Product: building block,

$\rightsquigarrow$ costs $E(n)$

Minimal polynomial: [Wiedemann 86]

$\rightsquigarrow$ iterative Krylov/Lanczos methods

$\rightsquigarrow O(nE(n) + n^2)$

Rank, Det, Solve: [Chen& Al. 02]

$\rightsquigarrow$ reduces to MinPoly + preconditioners

$\rightsquigarrow O\tilde{~}(nE(n) + n^2)$

Black box linear algebra

Matrix-Vector Product: building block,
$\rightsquigarrow$ costs $E(n)$

Minimal polynomial: [Wiedemann 86]
$\rightsquigarrow$ iterative Krylov/Lanczos methods
$\rightsquigarrow O(nE(n) + n^2)$

Rank, Det, Solve: [Chen& Al. 02]
$\rightsquigarrow$ reduces to MinPoly + preconditioners
$\rightsquigarrow \tilde{O}(nE(n) + n^2)$

Characteristic Poly.: [Dumas P. Saunders 09]
$\rightsquigarrow$ reduces to MinPoly, Rank, . . .

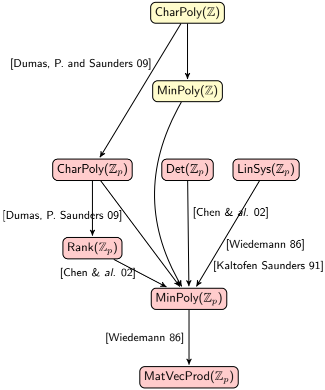
Black box linear algebra

Matrix-Vector Product: building block,
$\rightsquigarrow$ costs $E(n)$

Minimal polynomial: [Wiedemann 86]
$\rightsquigarrow$ iterative Krylov/Lanczos methods
$\rightsquigarrow O(nE(n) + n^2)$

Rank, Det, Solve: [Chen& Al. 02]
$\rightsquigarrow$ reduces to MinPoly + preconditioners
$\rightsquigarrow \tilde{O}(nE(n) + n^2)$

Characteristic Poly.: [Dumas P. Saunders 09]
$\rightsquigarrow$ reduces to MinPoly, Rank, ...



Outline

Reductions: linear algebra's arithmetic complexity

< 1969: $O(n^3)$ for everyone (Gauss, Householder, Danilevskĭi, etc)

Reductions: linear algebra's arithmetic complexity

< 1969: $O(n^3)$ for everyone (Gauss, Householder, Danilevskĭi, etc)

Matrix Product

[Strassen 69]:	$O(n^{2.807})$
⋮	
[Schönhage 81]	$O(n^{2.52})$
⋮	
[Coppersmith, Winograd 90]	$O(n^{2.375})$
⋮	
[Le Gall 14]	$O(n^{2.3728639})$

$\rightsquigarrow \mathsf{MM}(n) = O(n^\omega)$

Reductions: linear algebra's arithmetic complexity

< 1969: $O(n^3)$ for everyone (Gauss, Householder, Danilevskĭi, etc)

Matrix Product

[Strassen 69]: $O(n^{2.807})$

$\vdots$

[Schönhage 81] $O(n^{2.52})$

$\vdots$

[Coppersmith, Winograd 90] $O(n^{2.375})$

$\vdots$

[Le Gall 14] $O(n^{2.3728639})$

$\rightsquigarrow$ $\mathsf{MM}(n) = O(n^\omega)$

Other operations

[Strassen 69]: Inverse in $O(n^\omega)$

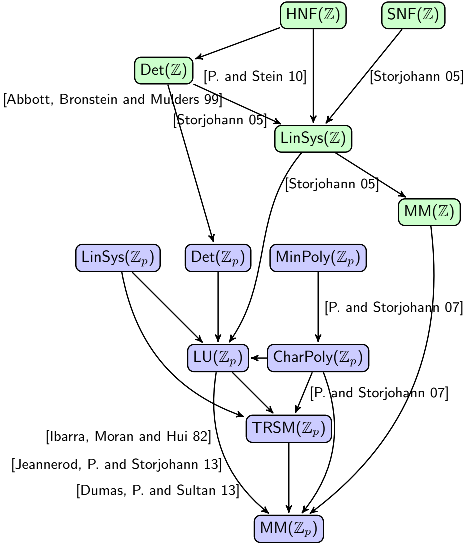
[Schönhage 72]: QR in $O(n^\omega)$

[Bunch, Hopcroft 74]: LU in $O(n^\omega)$

[Ibarra & *al.* 82]: Rank in $O(n^\omega)$

[Keller-Gehrig 85]: CharPoly in $O(n^\omega \log n)$

Reductions


HNF($\mathbb{Z}$)
SNF($\mathbb{Z}$)
Det($\mathbb{Z}$)
[P. and Stein 10]
[Storjohann 05]
[Abbott, Bronstein and Mulders 99]
[Storjohann 05]
LinSys($\mathbb{Z}$)
[Storjohann 05]
MM($\mathbb{Z}$)
LinSys($\mathbb{Z}_p$)
Det($\mathbb{Z}_p$)
MinPoly($\mathbb{Z}_p$)
[P. and Storjohann 07]
LU($\mathbb{Z}_p$)
CharPoly($\mathbb{Z}_p$)
[P. and Storjohann 07]
TRSM($\mathbb{Z}_p$)
[Ibarra, Moran and Hui 82]
[Jeannerod, P. and Storjohann 13]
[Dumas, P. and Sultan 13]
MM($\mathbb{Z}_p$)

Making theoretical reductions effective

Making theoretical reductions effective

Common mistrust

Fast linear algebra is

- ✗ never faster
- ✗ numerically unstable

Making theoretical reductions effective

Common mistrust

Fast linear algebra is

- ✗ never faster
- ✗ numerically unstable

Lucky coincidence

- ✓ building blocks **in theory** happen to be the most efficient routines **in practice**

$\rightsquigarrow$ reduction trees are still relevant

Making theoretical reductions effective

Common mistrust

Fast linear algebra is

- ✗ never faster
- ✗ numerically unstable

Lucky coincidence

- ✓ building blocks **in theory** happen to be the most efficient routines **in practice**

$\rightsquigarrow$ reduction trees are still relevant

Roadmap

1. Tune building blocks (MatMul)
2. Improve existing reductions (LU, Echelon)
   - ▷ leading constants
   - ▷ memory footprint
3. Produce new reduction schemes (CharPoly)

Matrix Multiplication over $\mathbb{Z}/p\mathbb{Z}$

Ingedients [Dumas, Gautier and P. 02]

- Compute over $\mathbb{Z}$ and delay modular reductions

$\rightsquigarrow$ $k\left(\frac{p-1}{2}\right)^2 < 2^{\text{mantissa}}$

Matrix Multiplication over $\mathbb{Z}/p\mathbb{Z}$

Ingedients [Dumas, Gautier and P. 02]

- Compute over $\mathbb{Z}$ and delay modular reductions

$$\rightsquigarrow \quad k\left(\frac{p-1}{2}\right)^2 < 2^{\text{mantissa}}$$

- Fastest integer arithmetic: `double`, `float` (SIMD and pipeline)
- Cache optimizations

$\rightsquigarrow$ numerical BLAS

Matrix Multiplication over $\mathbb{Z}/p\mathbb{Z}$

Ingedients [Dumas, Gautier and P. 02]

- Compute over $\mathbb{Z}$ and delay modular reductions

  $\rightsquigarrow 9^\ell \left\lfloor \frac{k}{2^\ell} \right\rfloor \left(\frac{p-1}{2}\right)^2 < 2^{\text{mantissa}}$
- Fastest integer arithmetic: `double`, `float` (SIMD and pipeline)
- Cache optimizations

  $\rightsquigarrow$ numerical BLAS
- Strassen-Winograd $6n^{2.807} + \dots$

Matrix Multiplication over $\mathbb{Z}/p\mathbb{Z}$

Ingedients [Dumas, Gautier and P. 02]

- Compute over $\mathbb{Z}$ and delay modular reductions

  $\rightsquigarrow$ $9^\ell \left\lfloor \frac{k}{2^\ell} \right\rfloor \left(\frac{p-1}{2}\right)^2 < 2^{\text{mantissa}}$

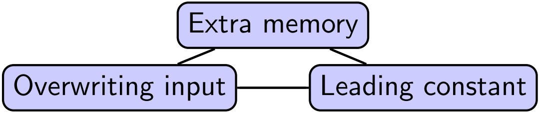
- Fastest integer arithmetic: `double`, `float` (SIMD and pipeline)
- Cache optimizations

  $\rightsquigarrow$ numerical BLAS

- Strassen-Winograd $6n^{2.807} + \dots$

with memory efficient schedules [Boyer, Dumas, P. and Zhou 09]

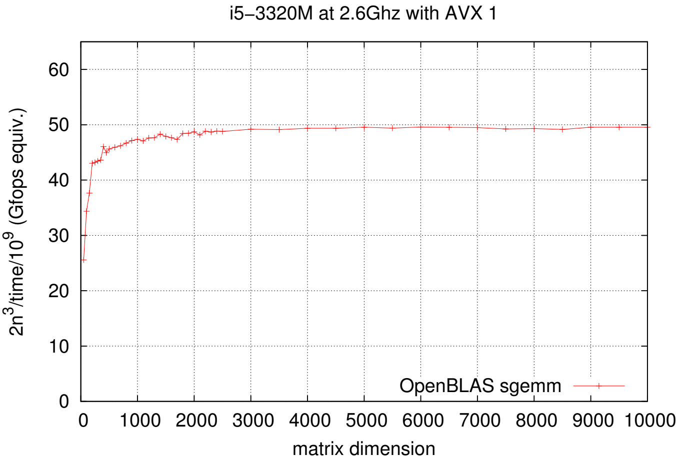
Tradeoffs:



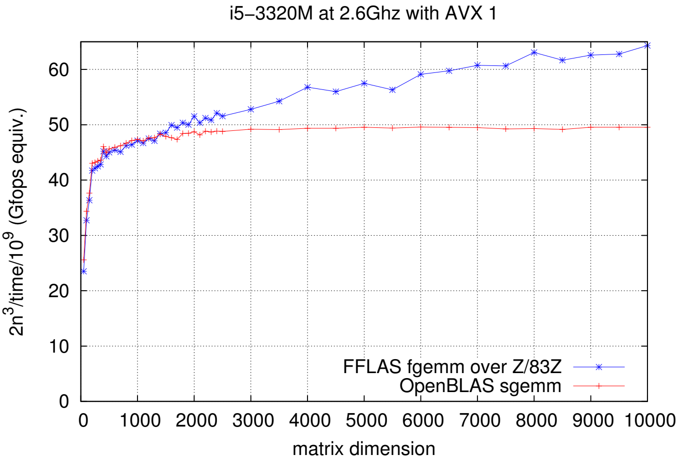
Fully in-place in

$7.2n^{2.807} + \dots$

Sequential Matrix Multiplication

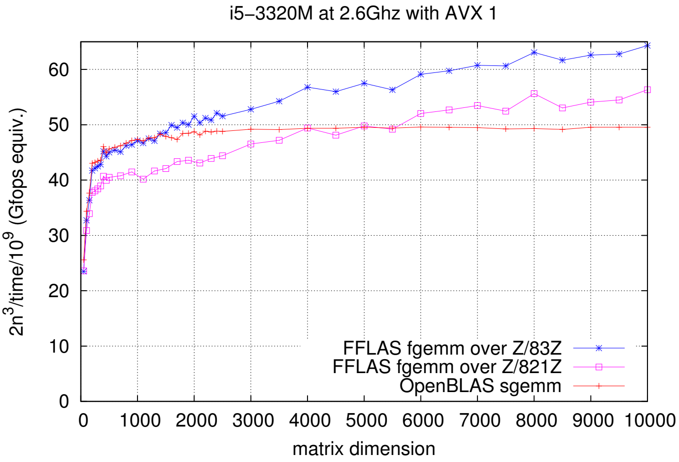


Sequential Matrix Multiplication



$p = 83$, $\rightsquigarrow$ 1 mod / 10000 mul.

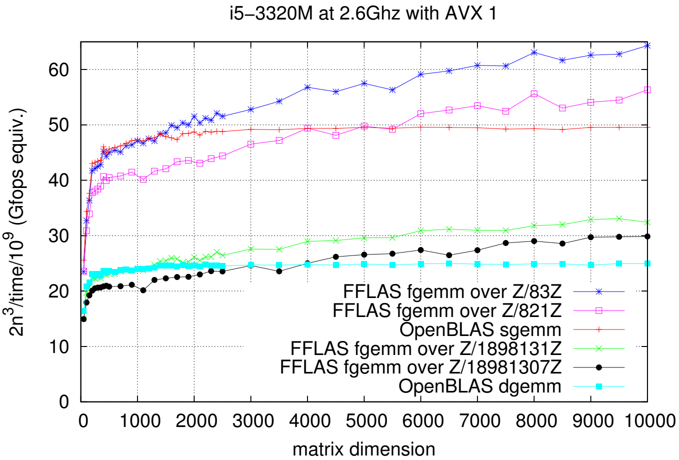
Sequential Matrix Multiplication



$p = 83$, $\rightsquigarrow$ 1 mod / 10000 mul.

$p = 821$, $\rightsquigarrow$ 1 mod / 100 mul.

Sequential Matrix Multiplication



$p = 83$, $\rightsquigarrow$ 1 mod / 10000 mul. $p = 1898131$, $\rightsquigarrow$ 1 mod / 10000 mul.

$p = 821$, $\rightsquigarrow$ 1 mod / 100 mul. $p = 18981307$, $\rightsquigarrow$ 1 mod / 100 mul.

Other routines

LU decomposition

- Block recursive algorithm $\rightsquigarrow$ reduces to MatMul $\rightsquigarrow$ $O(n^\omega)$

n	1000	5000	10000	15000	20000
`LAPACK-dgetrf`	**0.024s**	**2.01s**	**14.88s**	48.78s	113.66
`fflas-ffpack`	0.058s	2.46s	16.08s	**47.47s**	**105.96s**

Intel Haswell E3-1270 3.0Ghz using OpenBLAS-0.2.9

Other routines

LU decomposition

- Block recursive algorithm $\rightsquigarrow$ reduces to MatMul $\rightsquigarrow$ $O(n^\omega)$

n	1000	5000	10000	15000	20000
`LAPACK-dgetrf`	**0.024s**	**2.01s**	**14.88s**	48.78s	113.66
`fflas-ffpack`	0.058s	2.46s	16.08s	**47.47s**	**105.96s**

Intel Haswell E3-1270 3.0Ghz using OpenBLAS-0.2.9

Characteristic Polynomial

- A new reduction to matrix multiplication in $O(n^\omega)$.

n	1000	2000	5000	10000
`magma-v2.19-9`	1.38s	24.28s	332.7s	2497s
`fflas-ffpack`	**0.532s**	**2.936s**	**32.71s**	**219.2s**

Intel Ivy-Bridge i5-3320 2.6Ghz using OpenBLAS-0.2.9

Other routines

LU decomposition

- Block recursive algorithm $\rightsquigarrow$ reduces to MatMul $\rightsquigarrow O(n^\omega)$

n	1000	5000	10000	15000	20000
`LAPACK-dgetrf`	**0.024s**	**2.01s**	**14.88s**	48.78s	113.66
`fflas-ffpack`	0.058s	2.46s	16.08s	**47.47s**	**105.96s**

Intel Haswell E3-1270 3.0Ghz using OpenBLAS-0.2.9

$\times 7.63$ $\times 6.59$

Characteristic Polynomial

- A new reduction to matrix multiplication in $O(n^\omega)$.

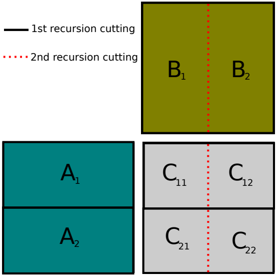
n	1000	2000	5000	10000
`magma-v2.19-9`	1.38s	24.28s	332.7s	2497s
`fflas-ffpack`	**0.532s**	**2.936s**	**32.71s**	**219.2s**

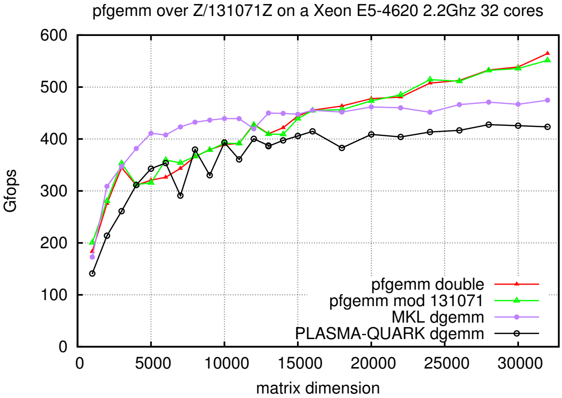
Intel Ivy-Bridge i5-3320 2.6Ghz using OpenBLAS-0.2.9

$\times 7.5$ $\times 6.7$

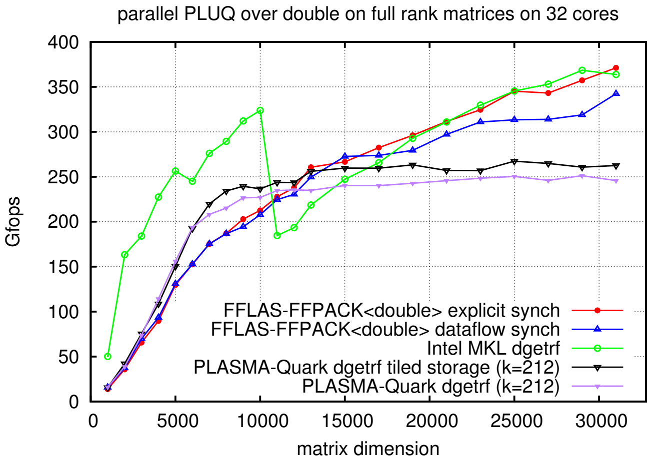
Outline

Parallel matrix multiplication





Gaussian elimination



Thank You.