Nemo: a computer algebra package for Julia

William Hart
TU Kaiserslautern

July 12, 2017

Introducing

A computer algebra package for the Julia programming language.

http://nemocas.org/

Background

2006 – David Harvey and I began the Flint C library for fast arithmetic

2010 – Fredrik Johansson, Sebastian Pancratz and I began Flint 2, complete rewrite of Flint

2013 – Claus Fieker and I began Antic C library for number field element arithmetic

2014 – Claus Fieker, Tommy Hofmann, Fredrik Johansson and I began Nemo generics

2015 – Oleksandr Motsak and I began Singular.jl package in Julia

2015 – Tommy Hofmann and Claus Fieker began Hecke algebraic number theory package

2016 – €7.6m grant for computer algebra, will fund OSCAR computer algebra system

New features in Flint

- LGPL license

New features in Flint

- LGPL license
- Exception system

New features in Flint

- LGPL license
- Exception system
- Quadratic sieve factorisation

New features in Flint

- LGPL license
- Exception system
- Quadratic sieve factorisation
- Elliptic curve factorisation

New features in Flint

- LGPL license
- Exception system
- Quadratic sieve factorisation
- Elliptic curve factorisation
- APRCL primality test

New features in Flint

- LGPL license
- Exception system
- Quadratic sieve factorisation
- Elliptic curve factorisation
- APRCL primality test
- Parallelised FFT

New features in Flint

- LGPL license
- Exception system
- Quadratic sieve factorisation
- Elliptic curve factorisation
- APRCL primality test
- Parallelised FFT
- Howell form

New features in Flint

- LGPL license
- Exception system
- Quadratic sieve factorisation
- Elliptic curve factorisation
- APRCL primality test
- Parallelised FFT
- Howell form
- Characteristic and minimal polynomial

New features in Flint

- LGPL license
- Exception system
- Quadratic sieve factorisation
- Elliptic curve factorisation
- APRCL primality test
- Parallelised FFT
- Howell form
- Characteristic and minimal polynomial
- van Hoeij factorisation for $\mathbb{Z}[x]$

New features in Flint

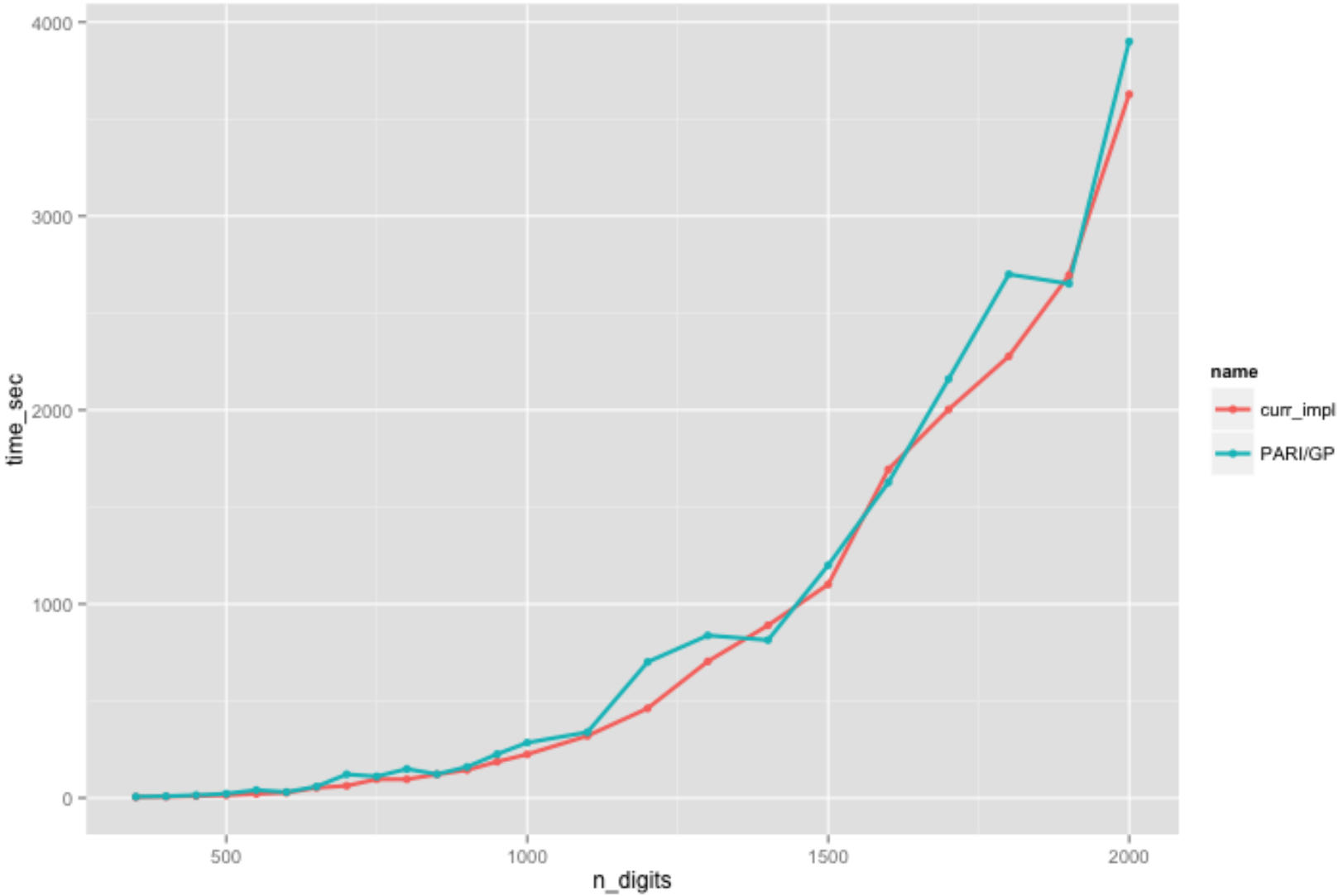
- LGPL license
- Exception system
- Quadratic sieve factorisation
- Elliptic curve factorisation
- APRCL primality test
- Parallelised FFT
- Howell form
- Characteristic and minimal polynomial
- van Hoeij factorisation for $\mathbb{Z}[x]$
- Multivariate polynomial arithmetic $\mathbb{Z}[x, y, z, \ldots]$

Integer factorisation : Quadratic sieve

Table : Quadratic sieve timings

Digits	Pari/GP	Flint (1 core)	Flint (4 cores)
50	0.43	0.55	0.39
59	3.8	3.0	1.7
68	38	21	14
77	257	140	52
83	2200	1500	540

APRCL primality test timings



FFT: Integer and polynomial multiplication

Table : FFT timings

Words	1 core	4 cores	8 cores
110k	0.07s	0.05s	0.05s
360k	0.3s	0.1	0.1s
1.3m	1.1s	0.4s	0.3s
4.6m	4.5s	1.5s	1.0s
26m	28s	9s	6s
120m	140s	48s	33s
500m	800s	240s	150s

Characteristic and minimal polynomial

Table : Charpoly and minpoly timings

Op	Sage 6.9	Pari 2.7.4	Magma 2.21-4	Giac 1.2.2	Flint
Charpoly	0.2s	0.6s	0.06s	0.06s	0.04s
Minpoly	0.07s	>160 hrs	0.05s	0.06s	0.04s

for 80×80 matrix over $\mathbb{Z}$ with entries in $[-20, 20]$ and minpoly of degree 40.

Multivariate multiplication

Table : "Dense" Fateman multiply bench

n	Sage	Singular	Magma	Giac	Flint
5	0.008s	0.001s	0.002s	0.0002s	0.0002s
10	0.56s	0.18s	0.12s	0.006s	0.004s
15	10s	5.6s	1.9s	0.11s	0.045s
20	76s	66s	16s	0.62s	0.50s
25	430s	410s	98s	2.8s	2.3s
30	1800s	1500s	440s	14s	10s

4 variables

Multivariate multiplication

Table : Sparse Pearce multiply bench

n	Sage	Singular	Magma	Giac	Flint
4	0.01s	0.003s	0.006s	0.004s	0.002s
6	0.20s	0.08s	0.08s	0.03s	0.02s
8	2.0s	1.4s	0.68s	0.28s	0.12s
10	11s	9.5s	3.0s	1.5s	0.55s
12	57s	38s	11s	4.8s	2.5s
14	210s	160s	37s	14s	11s
16	790s	510s	94s	39s	26s

5 variables

Exact quotient

Table : "Dense" quotient only

n	Sage	Singular	Magma	Giac	Flint
5	0.02s	0.003s	0.002s	0.0002s	0.0001s
10	1.1s	0.11s	0.16s	0.0039s	0.0022s
15	28s	1.5s	3.5s	0.049	0.022s
20	340s	19s	35s	0.25s	0.15s
25	2500s	130s	210s	1.1s	0.93s
30	——	470s	830s	6.0s	3.6s

4 variables

Exact quotient

Table : Sparse quotient only

n	Sage	Singular	Magma	Giac	Flint
4	0.49s	0.02s	0.005s	0.001s	0.0008s
6	107s	0.24s	0.17s	0.014s	0.010s
8	——	1.6s	3.1s	0.12s	0.068s
10	——	6.7s	27s	0.93s	0.34s
12	——	26s	140s	2.5s	1.7s
14	——	98s	630s	8.0s	6.7s
16	——	280s	2300s	22s	19s

5 variables

Divisibility testing

Table : "Dense" divisibility test with quotient

n	Sage	Singular	Magma	Giac	Flint
5	0.02s	0.006s	0.002s	0.001s	0.0005s
10	1.1s	0.56s	0.16s	0.05s	0.020s
15	28s	15s	3.3s	0.15s	0.054s
20	340s	150s	31s	0.90s	0.48s
25	2500s	840s	200s	4.1s	2.3s
30	——	3100s	830s	21s	11s

4 variables, returns quotient

Divisibility testing

Table : Sparse divisibility test with quotient

n	Sage	Singular	Magma	Giac	Flint
4	0.49s	0.03s	0.005s	0.002s	0.002s
6	107s	0.54s	0.17s	0.024s	0.021s
8	——	6.6s	3.1s	0.19s	0.15s
10	——	38s	27s	1.3s	0.69s
12	——	160s	140s	4.3s	2.8s
14	——	600s	630s	14s	9.0s
16	——	1900s	2300s	40s	26s

5 variables, returns quotient

Language for generic programming

In 2010 I started looking for a language for generic programming:

Language for generic programming

In 2010 I started looking for a language for generic programming:

- Support for Windows, Linux, Mac

Language for generic programming

In 2010 I started looking for a language for generic programming:

- Support for Windows, Linux, Mac
- 64 bit integers and double precision floats

Language for generic programming

In 2010 I started looking for a language for generic programming:

- Support for Windows, Linux, Mac
- 64 bit integers and double precision floats
- Console/REPL mode

Language for generic programming

In 2010 I started looking for a language for generic programming:

- Support for Windows, Linux, Mac
- 64 bit integers and double precision floats
- Console/REPL mode
- Operator overloading

Language for generic programming

In 2010 I started looking for a language for generic programming:

- Support for Windows, Linux, Mac
- 64 bit integers and double precision floats
- Console/REPL mode
- Operator overloading
- Fast generics and metaprogramming

Language for generic programming

In 2010 I started looking for a language for generic programming:

- Support for Windows, Linux, Mac
- 64 bit integers and double precision floats
- Console/REPL mode
- Operator overloading
- Fast generics and metaprogramming
- Maintained and popular

Language for generic programming

In 2010 I started looking for a language for generic programming:

- Support for Windows, Linux, Mac
- 64 bit integers and double precision floats
- Console/REPL mode
- Operator overloading
- Fast generics and metaprogramming
- Maintained and popular
- Open source

Language for generic programming

In 2010 I started looking for a language for generic programming:

- Support for Windows, Linux, Mac
- 64 bit integers and double precision floats
- Console/REPL mode
- Operator overloading
- Fast generics and metaprogramming
- Maintained and popular
- Open source
- Imperative syntax

Language for generic programming

In 2010 I started looking for a language for generic programming:

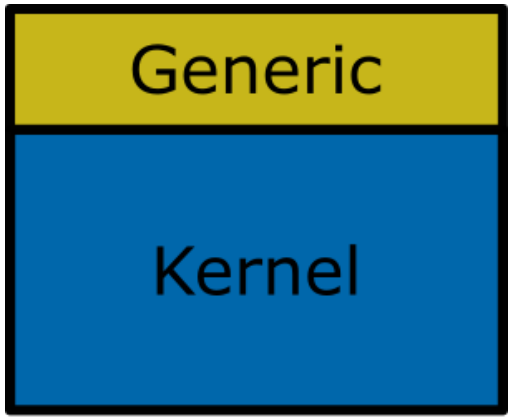
- Support for Windows, Linux, Mac
- 64 bit integers and double precision floats
- Console/REPL mode
- Operator overloading
- Fast generics and metaprogramming
- Maintained and popular
- Open source
- Imperative syntax
- Garbage collected

Language for generic programming

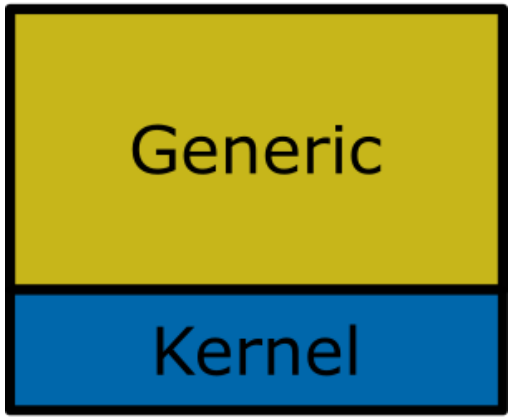
In 2010 I started looking for a language for generic programming:

- Support for Windows, Linux, Mac
- 64 bit integers and double precision floats
- Console/REPL mode
- Operator overloading
- Fast generics and metaprogramming
- Maintained and popular
- Open source
- Imperative syntax
- Garbage collected
- Easy/efficient C interop

Efficient generics

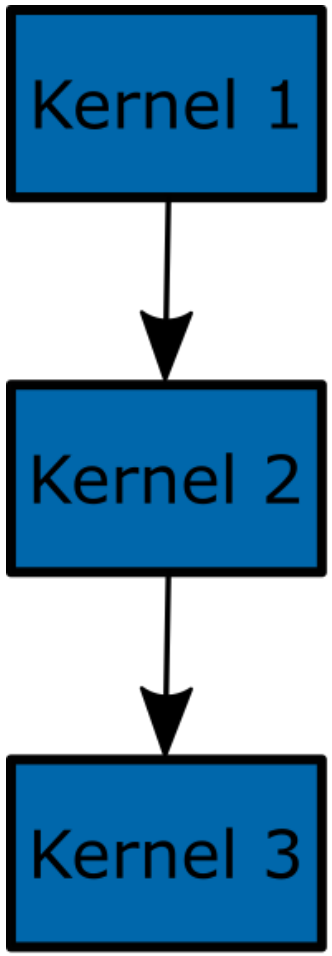


Fast generics

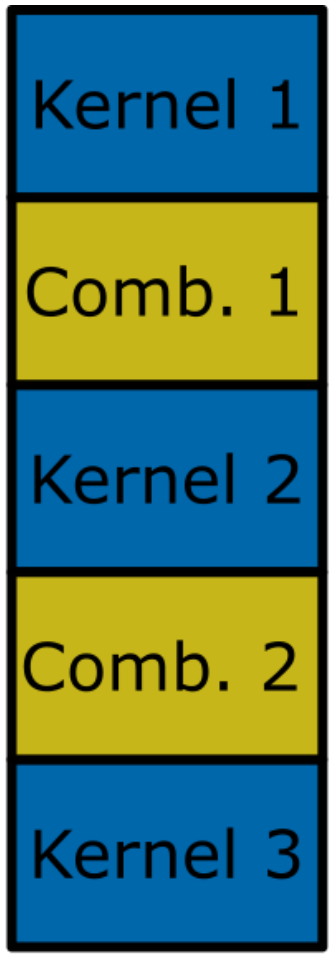


Slow generics

Efficient generics



Fast data transform



Generic bottleneck


julia

- JIT compilation : near C performance.
- Designed by mathematically minded people.
- Open Source (MIT License).
- Actively developed since 2009.
- Supports Windows, OSX, Linux, BSD.
- Friendly C/Python-like (imperative) syntax.

Julia is polymorphic:

```
gcd(a::Int, b::Int)
gcd(a::BigInt, b::BigInt)
gcd{T <: Field}(a::Poly{T}, b::Poly{T})
```

Julia is polymorphic:

```
gcd(a::Int, b::Int)
gcd(a::BigInt, b::BigInt)
gcd{T <: Field}(a::Poly{T}, b::Poly{T})
```

Julia supports multimethods:

```
*(a::Int, b::Matrix{Int})
*(a::Matrix{Int}, b::Int)
```

Julia is polymorphic:

```
gcd(a::Int, b::Int)
gcd(a::BigInt, b::BigInt)
gcd{T <: Field}(a::Poly{T}, b::Poly{T})
```

Julia supports multimethods:

```
*(a::Int, b::Matrix{Int})
*(a::Matrix{Int}, b::Int)
```

Julia supports triangular dispatch:

```
*{T <: QuotientRing, S <: Poly{T}}(x::T, y::S)
```

Julia supports coercion in a natural way:

```
+{T <: Domain}(a::Laurent{T}, b::Series{FractionField{T}})
```

Julia supports coercion in a natural way:

```
+{T <: Domain}(a::Laurent{T}, b::Series{FractionField{T}})
```

Julia supports:

- Custom array indexing
- Custom printing of objects
- Custom promotion rules and conversions

Julia supports coercion in a natural way:

```
+{T <: Domain}(a::Laurent{T}, b::Series{FractionField{T}})
```

Julia supports:

- Custom array indexing
- Custom printing of objects
- Custom promotion rules and conversions

Coming soon in Julia:

- Traits


NemO

Interfaces to C libraries:

- Flint

Interfaces to C libraries:

- Flint
- Arb

Interfaces to C libraries:

- Flint
- Arb
- Antic

Interfaces to C libraries:

- Flint
- Arb
- Antic
- Singular kernel (via Singular.jl)

Interfaces to C libraries:

- Flint
- Arb
- Antic
- Singular kernel (via Singular.jl)

Generic algorithms:

- Residue rings

Interfaces to C libraries:

- Flint
- Arb
- Antic
- Singular kernel (via Singular.jl)

Generic algorithms:

- Residue rings
- Fraction fields

Interfaces to C libraries:

- Flint
- Arb
- Antic
- Singular kernel (via Singular.jl)

Generic algorithms:

- Residue rings
- Fraction fields
- Dense univariate polynomials

Interfaces to C libraries:

- Flint
- Arb
- Antic
- Singular kernel (via Singular.jl)

Generic algorithms:

- Residue rings
- Fraction fields
- Dense univariate polynomials
- Sparse distributed multivariate polynomials

Interfaces to C libraries:

- Flint
- Arb
- Antic
- Singular kernel (via Singular.jl)

Generic algorithms:

- Residue rings
- Fraction fields
- Dense univariate polynomials
- Sparse distributed multivariate polynomials
- Dense linear algebra

Interfaces to C libraries:

- Flint
- Arb
- Antic
- Singular kernel (via Singular.jl)

Generic algorithms:

- Residue rings
- Fraction fields
- Dense univariate polynomials
- Sparse distributed multivariate polynomials
- Dense linear algebra
- Power series : absolute and relative

Interfaces to C libraries:

- Flint
- Arb
- Antic
- Singular kernel (via Singular.jl)

Generic algorithms:

- Residue rings
- Fraction fields
- Dense univariate polynomials
- Sparse distributed multivariate polynomials
- Dense linear algebra
- Power series : absolute and relative
- Permutation groups

Hecke

http://www.github.com/thofma/Hecke.git

- Orders and ideals in absolute number fields
- Fast ideal and element arithmetic in absolute number fields
- Verified computations with approximations using interval arithmetic whenever necessary (e.g. computation with embeddings or residue computation of Dedekind zeta functions)
- Sparse linear algebra over $\mathbf{Z}$
- Class and unit group computation
- Pseudo-Hermite normal form for modules over Dedekind domains
- Beginnings of class field theory and relative extensions

Future plans:

- Global function fields
- Class formations and more Galois cohomology
- Galois module structure
- Noncommutative algebras and orders

Thank You

http://nemocas.org/