Workpackage 5:
High Performance Mathematical Computing

Clément Pernet

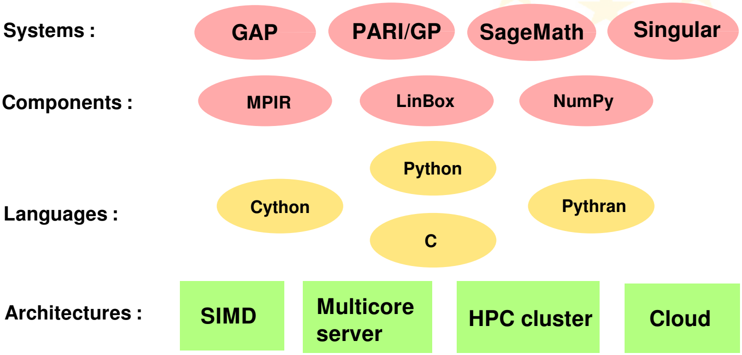
First OpenDreamKit Project review

Brussels, April 25, 2017

Goal: delivering high performance to maths users

Harnessing modern hardware $\rightsquigarrow$ parallelisation

- in-core parallelism (SIMD vectorisation)
- multi-core parallelism
- distributed computing: clusters, cloud



Goal: delivering high performance to maths users

Languages

- Computational Maths software use high level languages (e.g. Python)
- High performance delivered by languages close to the metal (C, assembly)

⇝ compilation, automated optimisation

Systems :	GAP	PARI/GP	SageMath	Singular
Components :	MPIR	LinBox	NumPy	
Languages :	Cython	Python, C	Pythran	
Architectures :	SIMD	Multicore server	HPC cluster	Cloud

Outline

Main tasks under review for the period

- Task 5.4: Singular
- Task 5.5: MPIR
- Task 5.6: Combinatorics
- Task 5.7: Pythran
- Task 5.8: SunGridEngine in JupyterHub

Progress report on other tasks

Task 5.4: Singular

Singular: A computer algebra system for polynomial computations.

- Already has a generic parallelization framework
- Focus on optimizing kernel routines for fine grain parallelism

D5.6: Quadratic sieving for integer factorization

D5.7: Parallelization of matrix fast Fourier Transform

D5.6: Quadratic Sieving for integer factorization

Quadratic Sieving for integer factorization

Problem: Factor an integer n into prime factors

Role: Crucial in algebraic number theory, arithmetic geometry.

Earlier status: no HPC implementation for large instances:

- only fast code for up to 17 digits,
- only partial sequential implementation for large numbers

D5.6: Quadratic Sieving for integer factorization

Achievements

- Completed and debugged implementation of large prime variant
- Parallelised sieving component of implementation using OpenMP
- Experimented with a parallel implementation of Block Wiedemann

Results

- Now modern, robust, parallel code for numbers in 17–90 digit range

D5.6: Quadratic Sieving for integer factorization

Achievements

- Completed and debugged implementation of large prime variant
- Parallelised sieving component of implementation using OpenMP
- Experimented with a parallel implementation of Block Wiedemann

Results

- Now modern, robust, parallel code for numbers in 17–90 digit range
- Significantly faster on small multicore machines

Table: Speedup for four cores (c/f single core):

Digits	50	60	70	80	90
Speedup	1.1×	1.76×	1.55×	2.69×	2.80×

D5.7: Parallelise and assembly optimize FFT

FFT: Fast Fourier Transform over $\mathbb{Z}/p\mathbb{Z}$

- Among the top 10 most important algorithms
- Key to fast arithmetic (integers, polynomials)
- Difficult to optimize: high memory bandwidth requirement

Earlier status:

- world leading **sequential** code in MPIR and FLINT;
- no parallel code.

D5.7: Parallelise and assembly optimize FFT

Achievements

- Parallelised Matrix Fourier implementation using OpenMP
- Assembly optimised butterfly operations in MPIR

Results:

- $\approx$ 15% speedup on Intel Haswell
- $\approx$ 20% speedup on Intel Skylake
- Significant speedups on multicore machines

Table: Speedup of large integer multiplication on 4/8 cores:

Digits	3M	10M	35M	125M	700M	3.3B	14B
4 cores	1.35×	2.67×	2.92×	2.92×	3.01×	2.95×	3.32×
8 cores	1.35×	3.56×	4.22×	4.36×	4.50×	4.31×	5.49×

Task 5.5: MPIR

MPIR : a library for big integer arithmetic

- Bignum operations: fundamental across all of computer algebra

D5.5: Assembly superoptimization

- MPIR contains assembly language routines for bignum operations
  $\rightsquigarrow$ hand optimised for every new microprocessor architecture
  $\rightsquigarrow \approx 3-6$ months of work for each architecture
- Superoptimisation: rearranges instructions to get optimal ordering

Earlier status:

- No assembly code for recent ($>$ 2012) Intel and AMD chips (Bulldozer, Haswell, Skylake, . . .)

D5.5: Assembly superoptimisation

Achievements

- A new assembly superoptimiser supporting recent instruction sets
- Superoptimised handwritten assembly code for Haswell and Skylake
- Hand picked faster assembly code for Bulldozer from existing implementations

Results:

- Sped up basic arithmetic operations for Bulldozer, Skylake and Haswell
- Noticeable speedups for bignum arithmetic for all size ranges

Op	Mul (s)	Mul (m)	Mul (b)	GCD (s)	GCD (m)	GCD (b)
Haswell	1.18×	1.27×	1.29×	0.72×	1.45×	1.27×
Skylake	1.15×	1.20×	1.22×	0.84×	1.65×	1.32×

s = 512 bits, m = 8192 bits, big = 100K bits

Task 5.6: Combinatorics

Perform a map/reduce operation on a very large set described recursively.

Large range of intensive applications in combinatorics:

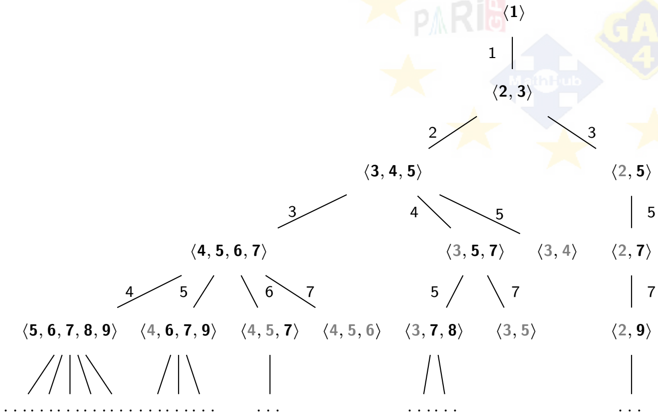
- Compute the cardinality, or more generally any kind of generating series
- Test a conjecture: i.e. find an element of S satisfying a specific property
- Count/list the elements of S having this property

Specificity of combinatorics:

- Typically the sets *don't fit in the computers* memory / disks and are enumerated on the fly (example of value: 10^{17} bytes).
- Easy to parallelize, if the set is flat (a list, a file, stored on a disk).

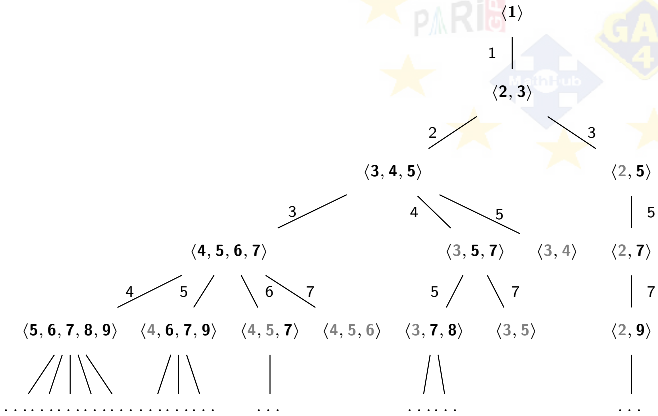
A Challenge: The tree of numerical semigroups

Extremely unbalanced. Need for an **efficient load balancing algorithm**.



A Challenge: The tree of numerical semigroups

Extremely unbalanced. Need for an **efficient load balancing algorithm**.



$\rightsquigarrow$ need for a high level task parallelization framework.

Work-Stealing System Architecture

A Python implementation

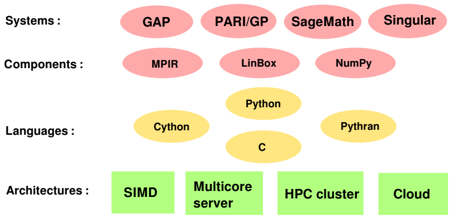
- Work stealing algorithm (Leiserson-Blumofe / Cilk)
- Easy to use, easy to call from sage
- Already, a dozen use case
- Scale well with the number of CPU cores
- Reasonably efficient (knowing that this is Python code).

References

- Trac Ticket 13580 `http://trac.sagemath.org/ticket/13580`
- *Exploring the Tree of Numerical Semigroups* Jean Fromentin and Florent Hivert `https://hal.inria.fr/UNIV-ROUEN/hal-00823339v3`

Task 5.7: Pythran

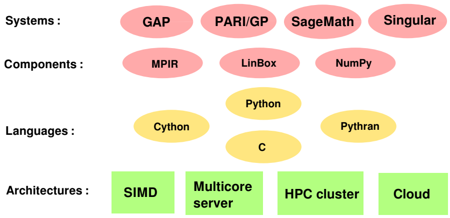
Pythran: a NumPy-centric Python to C compiler



- Many High level VRE rely on the Python language
- High performance is most often achieved by the C language

Task 5.7: Pythran

Pythran: a NumPy-centric Python to C compiler



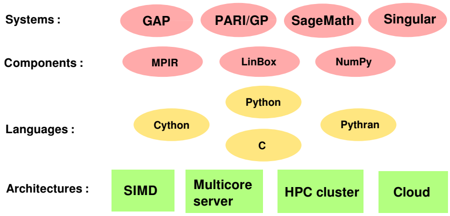
- Many High level VRE rely on the Python language
- High performance is most often achieved by the C language
- Python to C compilers:

  Cython: general purpose

  Pythran: narrower scope, better at optimizing Numpy code (Linear algebra)

Task 5.7: Pythran

Pythran: a NumPy-centric Python to C compiler



- Many High level VRE rely on the Python language
- High performance is most often achieved by the C language
- Python to C compilers:

  Cython: general purpose

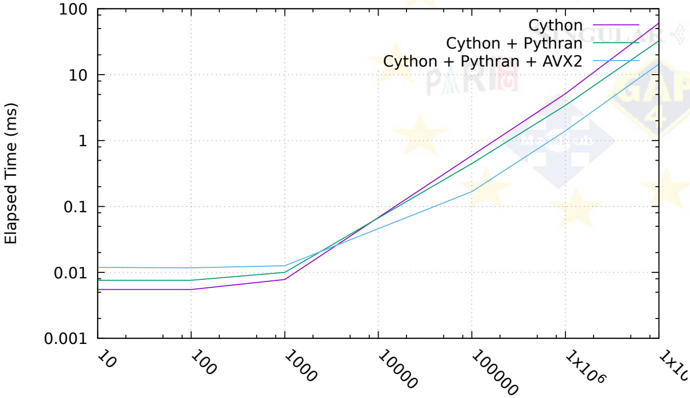
  Pythran: narrower scope, better at optimizing Numpy code (Linear algebra)

Goal: Implement the convergence

D5.4 Improve Pythran typing system

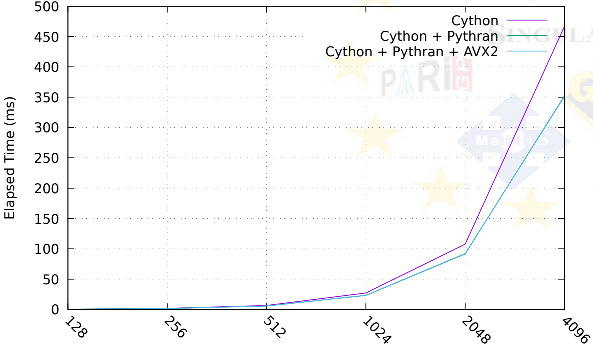
D5.2 Make Cython use Pythran backend to optimize Numpy code

D5.2: Make Cython use Pythran backend for NumPy code




```
import numpy
cimport numpy
def float_comp (numpy.ndarray[numpy.float_t , ndim=1] a,
                numpy.ndarray[numpy.float_t , ndim=1] b):
    return numpy.sqrt(numpy.sqrt(a*a+b*b))
```

D5.2: Make Cython use Pythran backend for NumPy code




```
def harris(numpy.ndarray[numpy.float_t, ndim=2] I):
  cdef int m = I.shape[0]
  cdef int n = I.shape[1]
  cdef numpy.ndarray[numpy.float_t,ndim=2] dx = (I[1:,:] - I[:m-1,:])[:,1:]
  cdef numpy.ndarray[numpy.float_t,ndim=2] dy = (I[:,1:] - I[:,:n-1])[1:,:]
  cdef numpy.ndarray[numpy.float_t, ndim=2] A = dx * dx
  cdef numpy.ndarray[numpy.float_t, ndim=2] B = dy * dy
  cdef numpy.ndarray[numpy.float_t, ndim=2] C = dx * dy
  cdef numpy.ndarray[numpy.float_t, ndim=2] tr = A + B
  cdef numpy.ndarray[numpy.float_t, ndim=2] det = A * B - C * C
  return det - tr * tr
```

Task 5.8: SunGridEngine integration in JupytherHub

Access to big compute

- Traditional access to supercomputers is difficult
- Notebooks are easy but run on laptops or desktops
- We need a way to connect notebooks to supercomputers

Sun Grid Engine

A job scheduler for Academic HPC Clusters

- Controls how resources are allocated to researchers
- One of the most popular schedulers

Achievements: D5.3

- Developed software to run Jupyter notebooks on supercomputers
- Users don't need to know details. They just log in.
- Demonstration install at University of Sheffield

Outline

Main tasks under review for the period

- Task 5.4: Singular
- Task 5.5: MPIR
- Task 5.6: Combinatorics
- Task 5.7: Pythran
- Task 5.8: SunGridEngine in JupyterHub

Progress report on other tasks

Progress report on other tasks

T5.1: Pari

- Generic parallelization engine is now mature, released (D5.10, due M24)

T5.2: GAP

- 6 releases were cut integrating contributions of D3.11 and D5.15
- Build system refactoring for integration of HPC GAP

T5.3: LinBox

- Algorithmic advances (5 articles) on linear algebra and verified computing
- Software releases and integration into SageMath