trip report

Workshop on interfacing math software and low level libraries

Enrico Guiraud, ROOT team

nthiery commented on Nov 23, 2017 • edited ▾ Contributor +😀

Workshop on interfacing (math) software with low level libraries

Dates: April 30th-May 4th
Location: Cernay-la-Ville, near Paris, France

nthiery commented on Nov 23, 2017 • edited ▾ Contributor +

Workshop on interfacing (math) software with low level libraries

Dates: April 30th-May 4th
Location: Cernay-la-Ville, near Paris, France

Math computational systems employ

- ***high level languages*** (e.g. Python) for expressivity, ease of use and prototyping
- ***low-level languages*** (e.g. C,C++) for performance, reusability

There are ***many existing different approaches to bind the two worlds together***

The workshop brought together developers to share their expertise on this topic

Sponsored by opendreamkit, an Horizon2020 project that funds development of computational math software

Links to program, participants, shared workshop notes etc. are available here

Who

~20 developers from
SageMath, CoCalc, pythran, cython, numba, QuantStack, DIANA, ROOT, ...
full list here

How

- very informal setting: a farmhouse in the countryside
- participants took turns cooking/serving food
- most people slept in shared rooms, one or two camped in the garden
- first round of per-project presentations
- then spontaneous formation of work groups, projects
- regular plenaries with project updates during the whole week

Sharing my experience: ROOT::RDataFrame

"low-to-high-level" development:

- build C++ library, make it easy to extend in C++

```
#include <ROOT/RDataFrame.hxx>
ROOT::EnableImplicitMT();
auto df = ROOT::MakeDataFrame("data/*.root");
auto rHist = tdf.Filter([](float x) { return x > .5; }, {"x"})
                .Define("r", sqrtSumSqr, {"x","y"})
                .Histo1D<float>("r");
```

full presentation here

Sharing my experience: ROOT::RDataFrame

"low-to-high-level" development:

- build C++ library, make it easy to extend in C++
- automatic generation of python bindings via pyROOT, cling

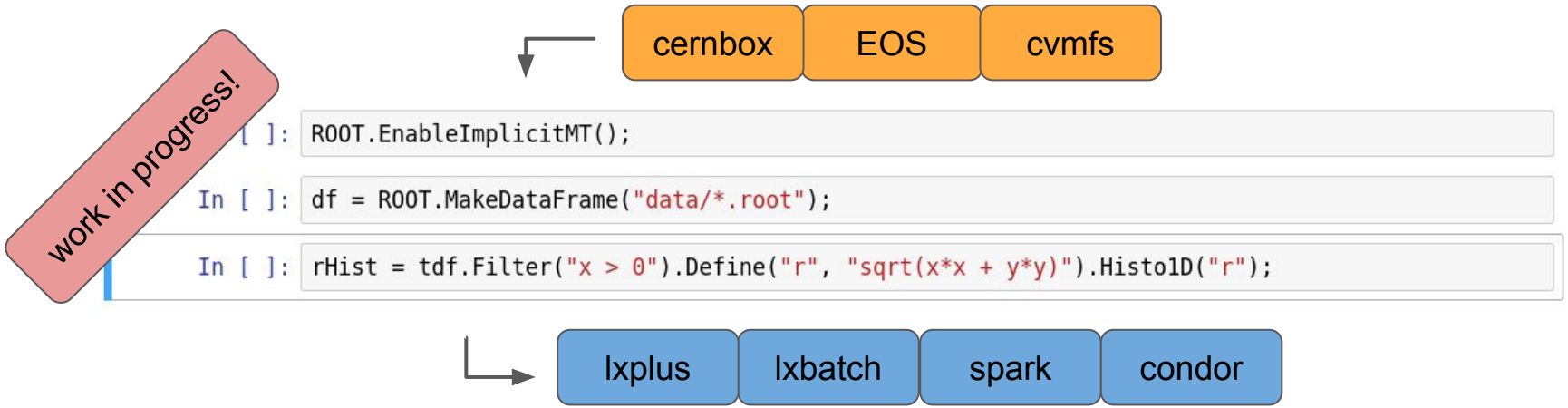
```
import ROOT
ROOT.EnableImplicitMT();
df = ROOT.MakeDataFrame("data/*.root")
rHist = tdf.Filter("x > .5")
           .Define("r", "sqrt(x*x + y*y)")
           .Histo1D("r")
```

full presentation here

Sharing my experience: ROOT::RDataFrame

"low-to-high-level" development:

- build C++ library, make it easy to extend in C++
- automatic generation of python bindings via pyROOT, cling
- augment user experience with SWAN notebooks



full presentation here

What the audience heard...

What the audience heard

AUTOMATIC GENERATION OF PYTHON BINDINGS

What the audience heard

AUTOMATIC GENERATION OF PYTHON BINDINGS

I ended up preparing
a tutorial on cppyy

It's publicly available here

Python bindings 1/3

numba

user writes: python
dev writes: python

- for python users and library developers who want C speed at zero cost
- jit subset of python (numpy, if, for, numerical computations) to LLVM IR to binary
- jit only when arguments are passed to the function to optimize for argument types
- generate C code that can be called from python code

```
@numba.jit(nopython=True, nogil=True)
def sum2d(arr):
    M, N = arr.shape
    result = 0.0
    for i in range(M):
        for j in range(N):
            result += arr[i,j]
    return result
```

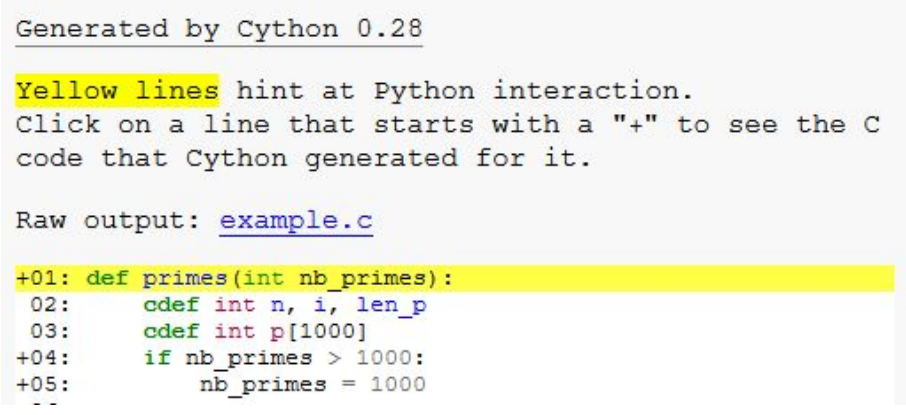
more on numba e.g. at this past DIANA topical meeting

Python bindings 2/3

cython

user writes: cython
dev writes: cython

- a programming language that looks like python with C type annotations
- a compiler for this language to C-with-python-bindings
- for python library developers who need to speed up part of them: compile functions to binary and expose them as C symbols
- for python users which need to speed-up hot loops that numba cannot handle
- for intermixing C (or a subset of C++) with python, when writing python
- can be used to call into existing C libraries



```
Generated by Cython 0.28

Yellow lines hint at Python interaction.
Click on a line that starts with a "+" to see the C
code that Cython generated for it.

Raw output: example.c

+01: def primes(int nb_primes):
 02:     cdef int n, i, len_p
 03:     cdef int p[1000]
+04:     if nb_primes > 1000:
+05:         nb_primes = 1000
```

Python bindings 3/3

cppyy

user writes: python
dev writes: C++

- for developers of C++ libraries that want to expose them from python
- automatic, lazy generation of bindings
- leverages cling's reflection to inspect C++ objects and callables and create their "equivalents" in python
- very fast thanks to usage of FFI rather than jit compilation

```
cppyy.cppdef("""
  struct Integer {
    Integer(int i) : num(i) {}
    int num;
    private:
    int p_num = -1;
  };
""")
from cppyy.gbl import Integer
m1 = Integer(42)
```

can also import .so or parse header

QuantStack: a modern C++ software stack for quantitative analysis

- recently founded startup of ~4 devs
- jokes about demo-driven development, but already a lot of meat
- offer a performant software stack in C++ and bindings in python, julia and R
- xtensor would become the lingua franca for multi-dimensional arrays in all these languages

loading data (in full or in batches) from whatever backend storage

in-memory data structure

xtensor: expression system to manipulate n-d arrays

high-level operations between labeled data

xframe: expression system to manipulate n-d tables (collections of labeled arrays)

interactive dev & viz

xeus-cling: c++ jupyter notebook

efficient computation

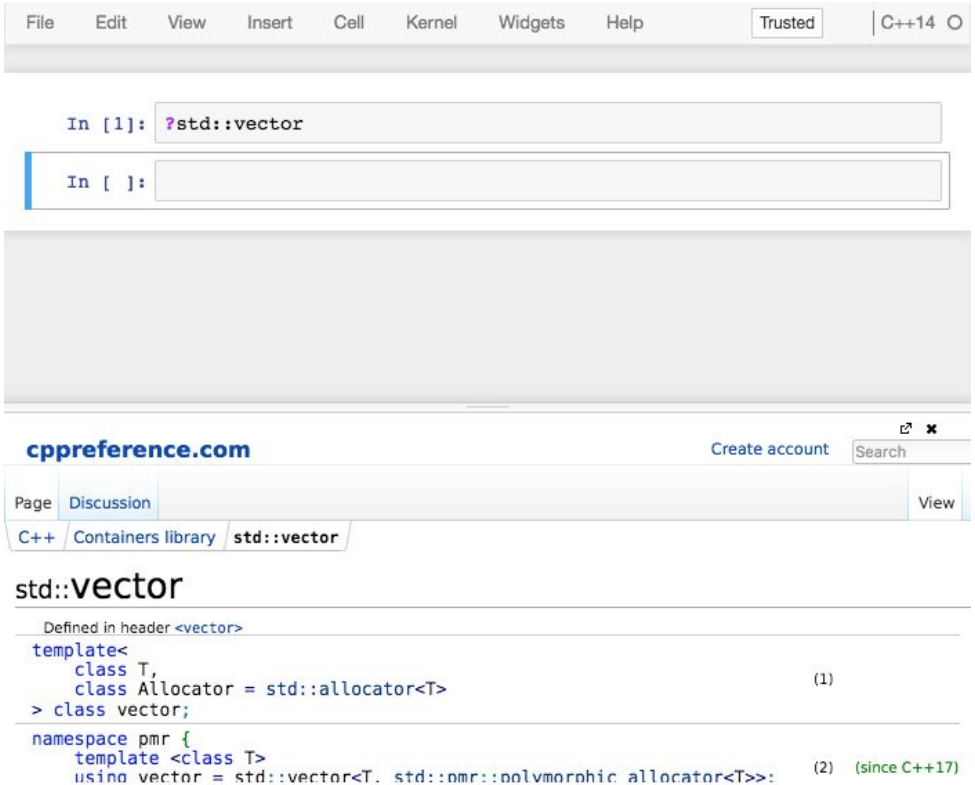
xsimd: vectorized algorithms

QuantStack: bits and pieces 1/2

xtensor: numpy-like C++ n-dimensional array

```cpp
xt::xarray<double> arr1
  {1.0, 2.0, 3.0};

xt::xarray<unsigned int> arr2
  {4, 5, 6, 7};

arr2.reshape({4, 1});

xt::xarray<double> res = xt::pow(arr1, arr2);

std::cout << res;

{{1, 16, 81},
 {1, 32, 243},
 {1, 64, 729},
 {1, 128, 2187}}
```

C++ jupyter kernel with widgets and live docs



QuantStack: bits and pieces 2/2

numpy to xtensor cheat-sheet

Python 3 - numpy	C++ 14 - xtensor
`np.sum(a, axis=[0, 1])`	`xt::sum(a, {0, 1})`
`np.sum(a)`	`xt::sum(a)`
`np.prod(a, axis=1)`	`xt::prod(a, {1})`
`np.prod(a)`	`xt::prod(a)`
`np.mean(a, axis=1)`	`xt::mean(a, {1})`
`np.mean(a)`	`xt::mean(a)`
`np.trapz(a, dx=2.0, axis=-1)  np.trapz(a, x=b, axis=-1)`	`xt::trapz(a, 2.0, -1)  xt::trapz(a, b, -1)`

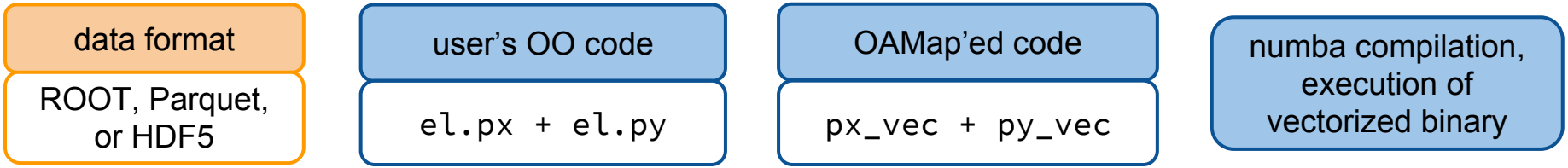
xleaflet widget for C++ jupyter kernel

```
In [*]: #include "xleaflet/xmap.hpp"
        #include "xleaflet/xtile_layer.hpp"
        #include "xleaflet/xwms_layer.hpp"
        #include "xleaflet/xsplit_map_control.hpp"

        auto map = xleaflet::map_generator()
            .center({50, 354})
            .zoom(5)
            .finalize();
        map

In [ ]: auto right_layer = xleaflet::tile_layer_generator()
            .url("https://mapl.vis.earthdata.nasa.gov/wmts-webmerc/MODIS_Terra_CorrectedReflectance_Tru
            .name("NASAGIBS.ModisTerraTrueColorCR")
            .attribution("Imagery provided by services from the Global Imagery Browse Services (GIBS),
            .max_zoom(9)
            .finalize();
        auto left_layer = xleaflet::tile_layer_generator()
            .url("https://cartodb-basemaps-{s}.global.ssl.fastly.net/light_all/{z}/{x}/{y}.png")
            .finalize();

In [ ]: auto control = xleaflet::split_map_control_generator()
            .left_layer(left_layer)
            .right_layer(right_layer)
            .finalize();
        map.add_control(control)

In [ ]:
```

OAMap

object ⇔ array mapping

- developed by Jim Pivarski as part of the DIANA project
- a generalization of the AOS ⇔ SOA concept
  and of ROOT's object splitting in columnar storage
- a programming model that lets users express their algorithms in a OO fashion
  and then transforms them in low-level, efficient, vectorized code
- the end result is a high-level analysis API
  which compiles down to very efficient vectorized data access



Conclusions

- very interesting to see what's out of our bubble in terms of scientific software (who knew, some people actually use Julia!)
- learned a lot on C, C++ python bindings
- world can benefit from our experience with unique technologies: cling, pyROOT, SWAN, …
- some interesting things out there (e.g. xtensor, xeus-cling): how can we benefit from them, as ROOT team and as EP-SFT in general?

Side-effects

- better template support in upstream cppyy
- QuantStack CEO is coming to the ROOT workshop
- project `libsemigroups` switched from cython to cppyy bindings
  - better support for function aliases in upstream cppyy
- CoCalc now includes support for ROOT v6.12