Computing in High Energy Physics 2015

Python in the Cling World

Wim T.L.P. Lavrijsen (WLavrijsen@lbl.gov)

A New Level of Integration between Python and C++

C++11/14 enhances interface specifications, clarifies resource ownership, and provides stricter guarantees. The Cling C++ interpreter brings a truly interactive experience and real dynamic behavior to the C++ language. Taken together, these developments allow PyROOT/cppyy and Cling to integrate the Python and C++ languages on a new level.

C++ becomes Pythonic!

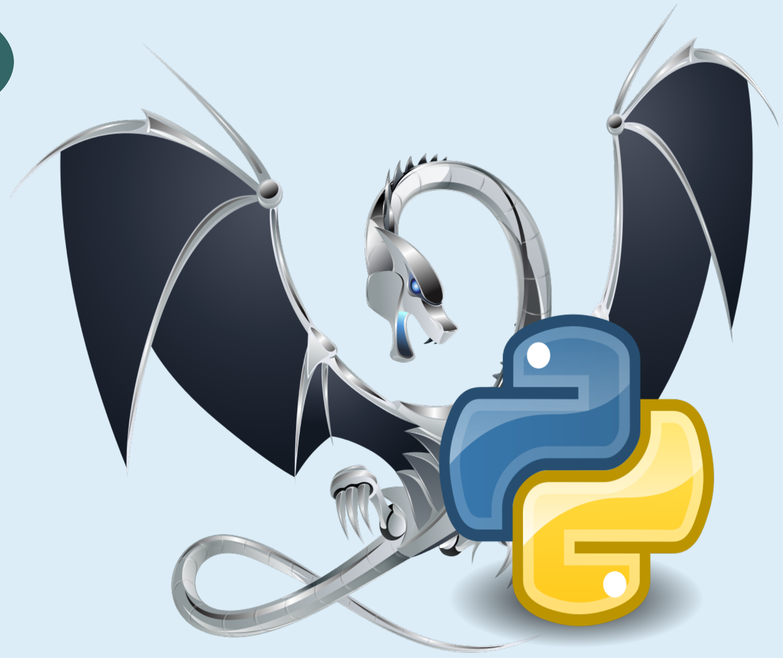
- “auto” typing
- range-based for loops
- std::tuple and std::tie
- uniform initializers
- lambda expressions
- standard algorithms
- parameter packs
- binary and raw literals

PyROOT Refactored

The bindings backend spun off as **cppyy**, with PyROOT adding ROOT-specifics on top of it.

An alternative, same API, cppyy module is available for PyPy.

cppyy is a drop-in replacement for PyCintex.



Dynamic Templates

```
>>> from ROOT import *
>>> gInterpreter.Declare( \
...  'template<typename T>\
...   struct Data {T fVal;};')
>>> results = std.map(\
...  std.string, Data(int))()
>>> d = Data(int)(); d.fVal = 42
>>> results['summary'] = d
>>> for tag, data in results:
...  print tag, ':', data.fVal
summary : 42
>>>
```

Cling can define new classes and instantiate templates, such as STL, with them. Templated methods are automatically derived from their arguments. Signature-based pythonizations are added on-the-fly.

Python from Cling

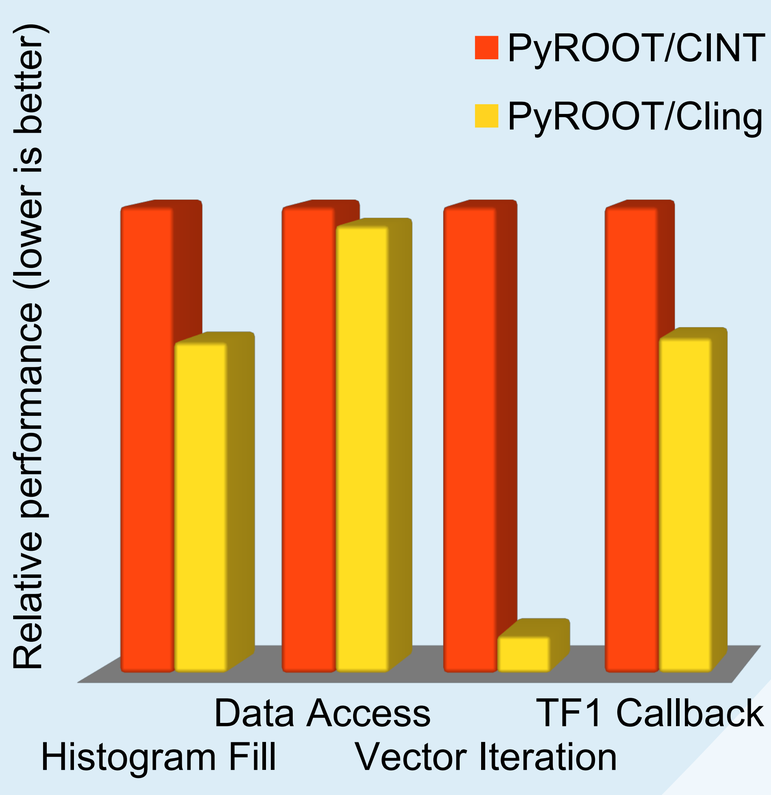
```
root[0] TPython::Import(“pymod”)
root[1] (double)pymod::func(3.14)
(double) 3.140000e+00
root[1] struct Aap :
    public pymod::PyKlass {
  Aap():fInt(42){}; int fInt; };
root[2] Aap a;
root[2] (char*)a.pass_it(“hello”)
(char *) “hello”
root[3] TPython::Prompt()
>>> a = ROOT.Aap()
>>> print a.fInt
42
```

Python classes and methods are accessible from the Cling prompt.

Free functions are supported.

Modules act as namespaces and simple casts convert types. New C++ classes can derive from Python ones and be passed back for normal use on either side.

Performance from Specialization



Increased correctness comes at a price, but that is more than negated by using lower-level functions. Data-member access by-passes Cling and the meta-layer, seeing little difference.

Specialization yields large benefits: type-specific iteration over STL vectors is >10x faster, and using pure function pointers removes overhead when fitting with Python callables.

Future Work

- On-going optimizations
- GIL safety and controlled multi-threading support
- NumPy integration
- Pythonization API
- Common cppyy backend for PyPy and CPython

References

http://doc.pypy.org/en/latest/cppyy.html
http://root.cern.ch/drupal/content/pyroot
http://root.cern.ch/drupal/content/pypyroot

CRD computational research division

U.S. DEPARTMENT OF ENERGY Office of Science