

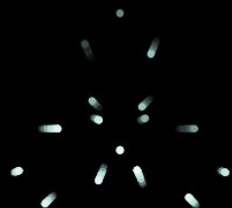


IRIS-HEP

Developing C++ modules support in CMSSW and Boost

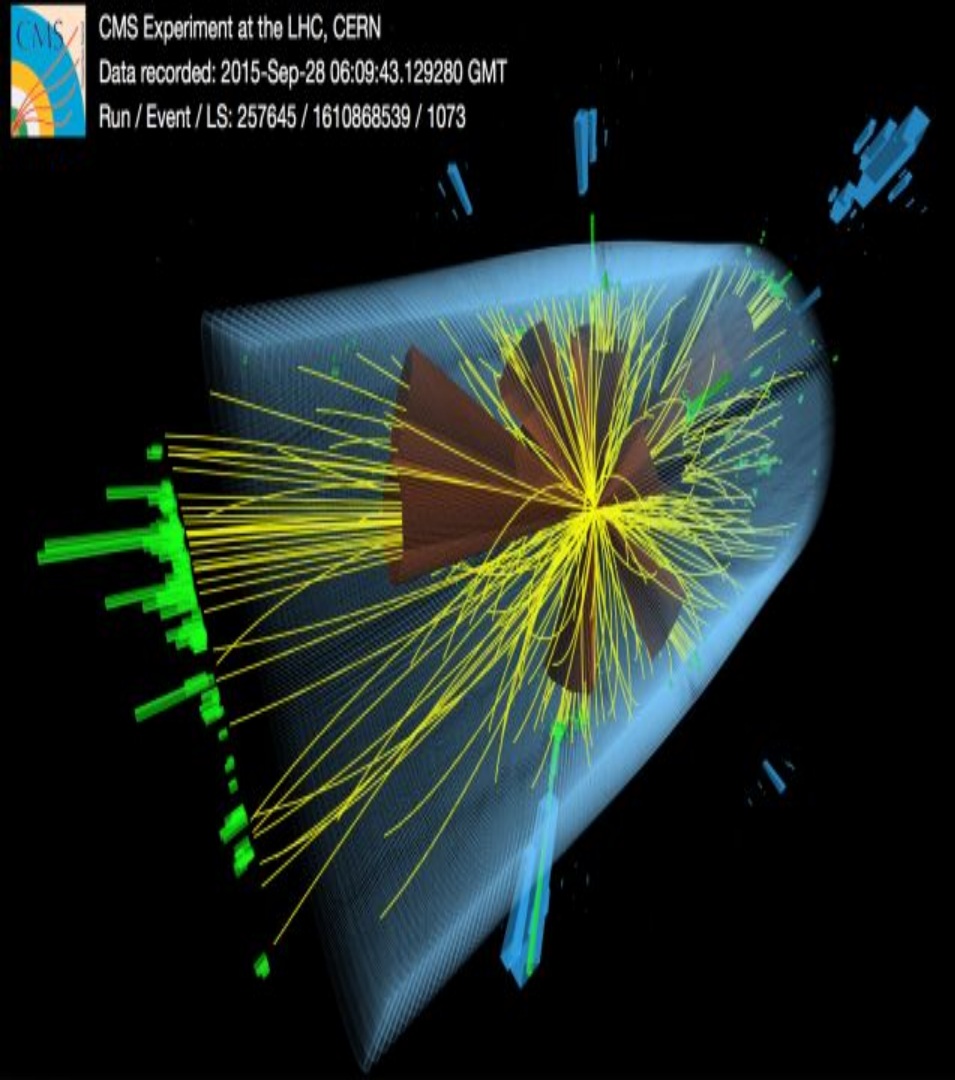
Mentors: Dr David Lange, Dr Vassil Vassilev

Student: Purva Chaudhari



What is CMSSW?

- CMS is a particle detector that is designed to see a wide range of particles and phenomena produced in high-energy collisions in the LHC.
- Many of CMS Software components (CMSSW) are hosted on Github.



About Project

Reduce Boost dependencies in
CMSSW

- Reducing boost dependencies helps us create more lightweight boost clang modules for upcoming c++20.
- This also reduces the amount of headers that we need to work on to be able to use c++20 clang modules.



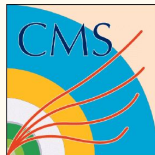
An abstract line drawing on the left side of the slide, featuring a dense network of thin, intersecting lines and a few small dots, resembling a complex web or a stylized tree structure.

Previous work done

- `boost::array` → `std::array`
- `boost::unordered_map` → `std::unordered_map`
- `boost::unordered_set` → `std::unordered_set`
- `boost::function` → `std::function`
- `boost::random` → `std`
- `boost::hash` → `std::hash`
- `boost::filesystem` → `std::filesystem`
- `boost::mutex` → `std::mutex`
- `boost::variant` → `std::variant`

My current contributions

- ❑ `boost::bind` → `std::bind`
- ❑ `boost::python` → `pybind11`
- ❑ `boost::lexical_cast` → corresponding stl casting methods
- ❑ Tried `boost::regex` → `std::regex` (could not due to lower performance of `std::regex`)
- ❑ Checked out probable replacements methods for `boost/string/algorithms`



boost::bind → std::bind

Fireworks/Core/test/unittest_changemanager.cc		<input type="checkbox"/> Viewed	...
@@ -12,7 +12,6 @@			
12		12	
13	// system include files	13	// system include files
14	#include <boost/test/auto_unit_test.hpp>	14	#include <boost/test/auto_unit_test.hpp>
15	- #include <boost/bind.hpp>		
16	#include <boost/test/test_tools.hpp>	15	#include <boost/test/test_tools.hpp>
17		16	
18	#include "TClass.h"	17	#include "TClass.h"
@@ -96,8 +95,8 @@ BOOST_AUTO_TEST_CASE(changemanager) {			
96	//NOTE: have to pass a pointer to the listener else the bind will	95	//NOTE: have to pass a pointer to the listener else the bind will
97	// create a copy of the listener and the original one will never	96	// create a copy of the listener and the original one will never
98	// 'hear' any signal	97	// 'hear' any signal
99	- item.changed_.connect(boost::bind(&Listener::listen, &listener, _1));	98	+ item.changed_.connect(std::bind(&Listener::listen, &listener, std::placeholders::_1));
100	- item.itemChanged_.connect(boost::bind(&ItemListener::listen, &listener, _1));	99	+ item.itemChanged_.connect(std::bind(&ItemListener::listen, &listener, std::placeholders::_1));
101		100	
102	BOOST_CHECK(listener.nHeard_ == 0);	101	BOOST_CHECK(listener.nHeard_ == 0);
103		102	

Boost::python → pybind11

1. Boost::python::list/dict → pybind11::list/dict

```
51
52 - boost::python::list PlotBase::inputParams() const {
53 -     boost::python::list tmp;
54     for (const auto& ip : m_inputParams) {
55         tmp.append(ip);
56     }
57     return tmp;
58 }
59
60 - void PlotBase::setInputParamValues(const boost::python::dict& values) {
```

```
>>
54 + py::list PlotBase::inputParams() const {
55 +     py::list tmp;
56     for (const auto& ip : m_inputParams) {
57         tmp.append(ip);
58     }
59     return tmp;
60 }
61
62 + void PlotBase::setInputParamValues(const py::dict& values) {
```

2. Boost::python::list/dict → object.cast

```
78 - boost::python::tuple entry = boost::python::extract<boost::python::tuple>
    (tagsWithTimeBoundaries[i]);
79 -     std::string tagName = boost::python::extract<std::string>(entry[0]);
80 -     std::string time0s = boost::python::extract<std::string>(entry[1]);
81 -     std::string time1s = boost::python::extract<std::string>(entry[2]);
82     cond::Time + time0 = boost::lexical_cast<cond::Time>(+time0s);
```

```
81 +     py::tuple entry = tagsWithTimeBoundaries[i].cast<py::tuple>();
82 +     std::string tagName = entry[0].cast<std::string>();
83 +     std::string time0s = entry[1].cast<std::string>();
84 +     std::string time1s = entry[2].cast<std::string>();
85     cond::Time + time0 = boost::lexical_cast<cond::Time>(+time0s);
```

Boost::python → pybind11

3. Boost python module → pybind11 module

CondCore/Utilities/plugins/Module_PayloadInspector.cc

... @@ -1,11 +1,17 @@

1 #include "CondCore/Utilities/interface/PayloadInspector.h"

2 - #include <boost/python.hpp>

3 - #include <boost/python/suite/indexing/vector_indexing_suite.hpp>

4

5 - BOOST_PYTHON_MODULE(pluginModule_PayloadInspector) {

6 - boost::python::class_<cond::payloadInspector::ModuleVersion>("ModuleVersion")

7 - .def_readonly("label", &cond::payloadInspector::ModuleVersion::label);

8 - boost::python::class_<cond::payloadInspector::PlotBase>("PlotBase")

1 #include "CondCore/Utilities/interface/PayloadInspector.h"

2 + #include <pybind11/pybind11.h>

3 + #include <pybind11/stl_bind.h>

4 + #include <pybind11/functional.h>

5 + #include <pybind11/stl.h>

6

7 + namespace py = pybind11;

8 +

9 + PYBIND11_MODULE(pluginModule_PayloadInspector, m) {

10 + py::class_<cond::payloadInspector::ModuleVersion,

11 + m, "ModuleVersion">(

12 + .def("label", []() { return cond::payloadInspector::ModuleVersion::label; });

13 +

Boost::lexical_cast → stl casting

1. Upcasting/downcasting → static_cast

6 CondCore/BeamSpotPlugins/test/testBeamSpotPayloadInspector.cpp	
@@ -27,8 +27,8 @@ int main(int argc, char** argv) {	
// BeamSpot	
std::string tag = "BeamSpotObjects_PCL_byLumi_v0_prompt";	
- cond::Time_t start = boost::lexical_cast<unsigned long long>(1406876667347162);	27 // BeamSpot
- //cond::Time_t end = boost::lexical_cast<unsigned long long>(1406876667347162);	28
	29 std::string tag = "BeamSpotObjects_PCL_byLumi_v0_prompt";
	30 + cond::Time_t start = static_cast<unsigned long long>(1406876667347162);
	31 + //cond::Time_t end = static_cast<unsigned long long>(1406876667347162);
	22

2. From string to int/double/unsigned long long → std::stoi/ std::stod/ std::stoull

4 CondCore/DBOutputService/src/OnlineDBOutputService.cc	
@@ -69,8 +69,8 @@ namespace cond {	
9 if (!std::getline(sinfo, slumi, ',')) {	
0 throw Exception("Can't get lumi id from OMS Service.");	
1 }	
2 - unsigned int run = boost::lexical_cast<unsigned int>(srun);	69 if (!std::getline(sinfo, slumi, ',')) {
3 - unsigned int lumi = boost::lexical_cast<unsigned int>(slumi);	70 throw Exception("Can't get lumi id from OMS Service.");
4 lastLumiProcessed = cond::time::lumiTime(run, lumi);	71 }
5 return lastLumiProcessed;	72 + unsigned int run = std::stoul(srun);
	73 + unsigned int lumi = std::stoul(slumi);
	74 lastLumiProcessed = cond::time::lumiTime(run, lumi);
	75 return lastLumiProcessed;

Boost::lexical_cast → stl casting

3. From int/ unsigned/ template to string → std::to_string

```
@@ -75,7 +74,7 @@ namespace eos {
```

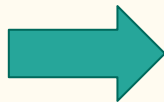
```
75     template <typename T>
76     portable_archive_exception(const T& abnormal)
77         : boost::archive::archive_exception(other_exception), msg("serialization of illegal
floating point value: ") {
78 -         msg += boost::lexical_cast<std::string>(abnormal);
79     }
```

```
74     template <typename T>
75     portable_archive_exception(const T& abnormal)
76         : boost::archive::archive_exception(other_exception), msg("serialization of illegal
floating point value: ") {
77 +         msg += std::to_string(abnormal);
78     }
```

Probable approach to eliminate boost algorithms

1. boost::algorithm::split

```
12 int main()
13 {
14     string input("Red,Green,Blue,Black,White,Orange,Purple,Yellow");
15     vector<string> result;
16     auto start = high_resolution_clock::now();
17
18     boost::split(result, input, boost::is_any_of(","));
19
20     auto stop = high_resolution_clock::now();
21     auto duration = duration_cast<microseconds>(stop - start);
22
23     cout << "Time taken by function: "
24           << duration.count() << " microseconds" << endl;
25
26     for (int i = 0; i < result.size(); i++)
27         cout << result[i] << endl;
28     return 0;
29 }
30
31 /*
32 OUTPUT:
33 Time taken by function: 30 microseconds
```



```
13 void tokenize(std::string const &str, const char delim,
14               std::vector<std::string> &out)
15 {
16     size_t start;
17     size_t end = 0;
18
19     while ((start = str.find_first_not_of(delim, end)) != std::string::npos)
20     {
21         end = str.find(delim, start);
22         out.push_back(str.substr(start, end - start));
23     }
24 }
25
26 int main()
27 {
28     vector<string> result;
29     std::string s = "Red,Green,Blue,Black,White,Orange,Purple,Yellow";
30     const char delim = ',';
31
32     std::vector<std::string> out;
33     auto start = high_resolution_clock::now();
34     tokenize(s, delim, out);
35     auto stop = high_resolution_clock::now();
36     auto duration = duration_cast<microseconds>(stop - start);
37     cout << "Time taken by function: "
38           << duration.count() << " microseconds" << endl;
39
40     for (auto &s: out) {
41         std::cout << s << std::endl;
42     }
43     return 0;
44 }
45 /*
46 OUTPUT:
47 Red
48 Green
49 Blue
50 Black
51 White
52 Orange
53 Purple
54 Yellow
55 Time taken by function: 11 microseconds
```

Existing boost dependencies

- `boost::regex` → `std::regex` has low performance
- `boost::format` → `std::format` c++20.
- `boost::posix` → `std::chrono` can be a probable approach
- `boost::serialization`
- `boost::iterator`
- `boost::spirit`
- `boost::algorithm`



Thank You
