

Faster Collections in RooFit

S. Hageboeck (CERN, EP-SFT) for the ROOT team

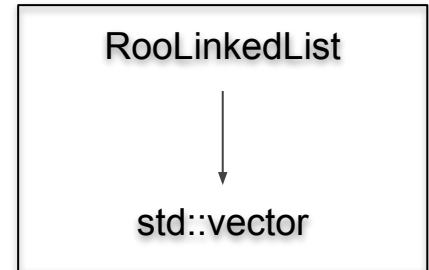
ROOT

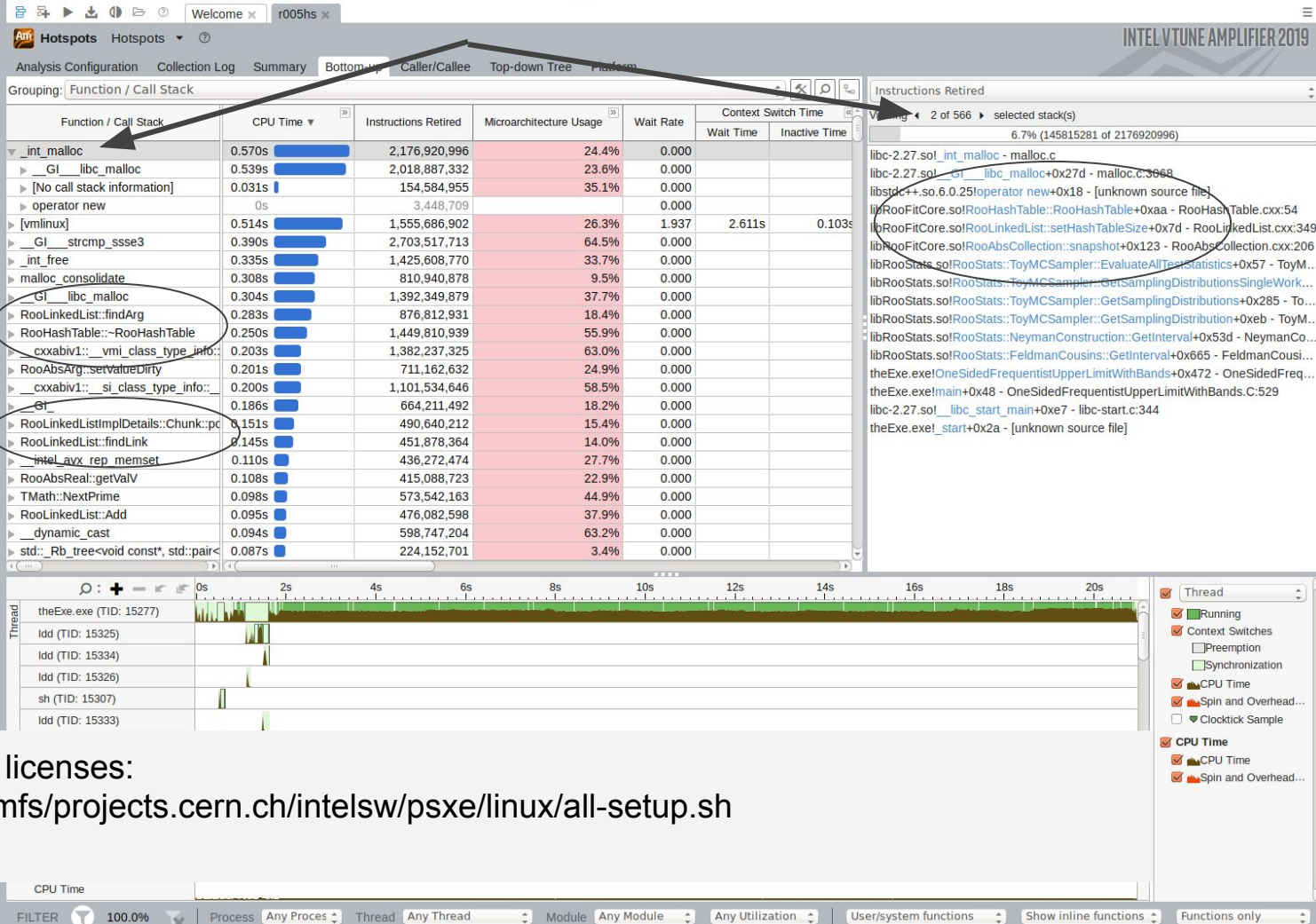
Data Analysis Framework

<https://root.cern>

Roofit has three kinds of collections:

- ▶ RooLinkedList:
 - Linked list of most general class in Roofit (RooAbsArg)
 - Memory pool for faster allocations & more data locality
 - Can enable a hash list to faster find elements by name
- ▶ RooArgSet:
 - Set of RooAbsArgs
 - RooLinkedList at backend
- ▶ RooArgList:
 - List of RooAbsArgs. Mostly used like a `std::vector`.
 - RooLinkedList at backend
- ▶ Most common operations (for all three):
 - **Forward iteration!!**
 - Search element by name or address (= forward iteration)
 - Add element (to back!)





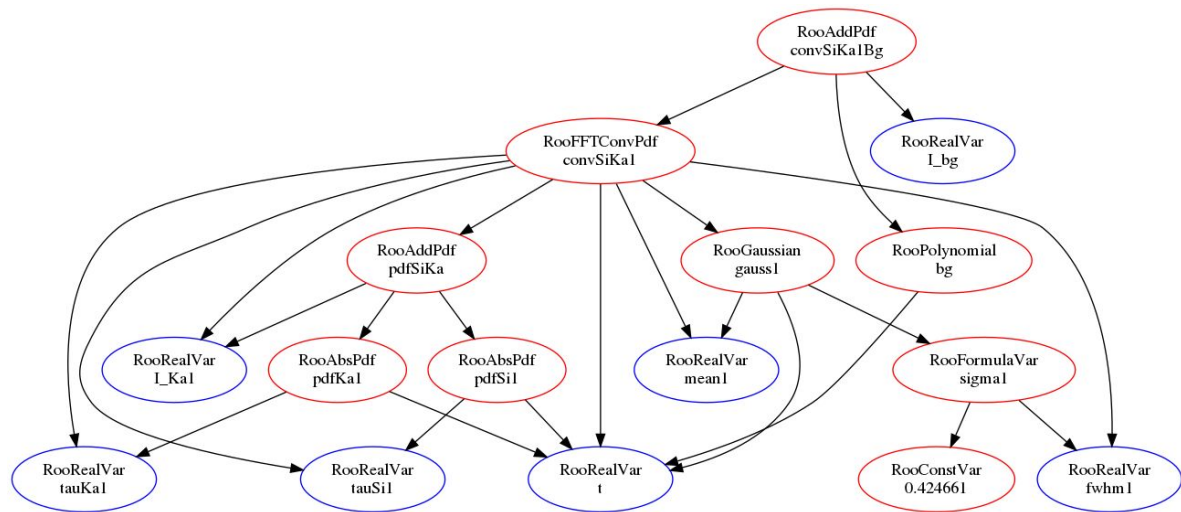
CERN has licenses:
 source /cvmfs/projects.cern.ch/intel/psxe/linux/all-setup.sh

Collections:

- ▶ Expression tree (+ almost everything in RooFit) stored as `RooLinkedList<RooAbsArg*>`
 - Often small search & iterates
- ▶ Toy MonteCarlo generation:
 - ~50% of L3 cache misses due to linked list + hash table operations

▶ The plan:

- Replace `LinkedList` by `std::vector`
- Provide STL-like interface





The Challenge

- ▶ Axel: “How much user code are you going to break?”
→ All ...
- ▶ The old collections directly expose the underlying storage implementation through the iterators

```
C Compare Viewer
Local: RooAbsCollection.h
131
132 const_iterator begin() const {
133     return _list.begin();
134 }
135
136 const_iterator end() const {
137     return _list.end();
138 }
139
140 Storage_t::size_type size() const {
141     return _list.size();
142 }
143
144 void reserve(Storage_t::size_type count) {
145     _list.reserve(count);
146 }
147
148 inline Int_t getSize() const {
149     // Return the number of elements in the collection
150     return _list.size();
151 }
152 ...

RooAbsCollection.h f84668d (Stephan Hageboeck)
91 RooAbsCollection* selectByAttrib(const char* name, Bool_t value) const ;
92 RooAbsCollection* selectCommon(const RooAbsCollection& refColl) const ;
93 RooAbsCollection* selectByName(const char* nameList, Bool_t verbose=kFALSE)
94 Bool_t equals(const RooAbsCollection& otherColl) const ;
95 Bool_t overlaps(const RooAbsCollection& otherColl) const ;
96
97 // export subset of TFlashList interface
98 inline TIterator* createIterator(Bool_t dir = kIterForward) const {
99     // Create and return an iterator over the elements in this collection
100     return _list.MakeIterator(dir);
101 }
102
103 RooLinkedListIter iterator(Bool_t dir = kIterForward) const ;
104 RooFIter fwdIterator() const { return RooFIter(&_list); }
105
106 inline Int_t getSize() const {
107     // Return the number of elements in the collection
108     return _list.GetSize();
109 }
110 inline RooAbsArg *first() const {
111     // Return the first element in this collection
112     return (RooAbsArg*) _list.First();
```



Solution

- ▶ Three kinds of old iterators need to be supported (all in use)
- ▶ RooLinkedList not completely gone anyway
- ▶ Write wrapper that delegates to RooLinkedList or STL as needed
- ▶ Downside: slower
 - Extra layer with virtual dispatch
 - Need to create&destroy iterators (and hand into userland) for polymorphy

```
//// Iterator for RooFIter-compatible iterators
class GenericRooFIter
{
public:
virtual RooAbsArg * next() = 0;
virtual ~GenericRooFIter() {}
};

////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
/// A one-time forward iterator working on RooLinkedList or RooAbsCollection.
/// This wrapper separates the interface visible to the outside from the actual
/// implementation of the iterator.
class RooFIter final
{
public:
RooFIter(std::unique_ptr<GenericRooFIter> && itImpl) : fIterImpl{std::move(itImpl)} {}
RooFIter(const RooFIter &) = delete;
RooFIter(RooFIter &&) = default;
RooFIter & operator=(const RooFIter &) = delete;
RooFIter & operator=(RooFIter &&) = default;

RooAbsArg *next() {
return fIterImpl->next();
}

private:
std::unique_ptr<GenericRooFIter> fIterImpl;
};

////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
/// Implementation of a GenericRooFIter for the RooLinkedList
class RooFIterForLinkedList final : public GenericRooFIter
{
public:
RooFIterForLinkedList() {}
RooFIterForLinkedList(const RooLinkedList* list) : fPtr (list->_first) {}

/// Return next element in collection
RooAbsArg *next() override {
if (!fPtr) return nullptr ;
TObject* arg = fPtr->_arg ;
fPtr = fPtr->_next;
return (RooAbsArg*) arg ;
}

private:
const RooLinkedListElem * fPtr{nullptr}; ///! Next link element
};
```



The Challenge II

- ▶ RooLinkedList:
 - Remove/add/replace before and after current iterator
 - No reallocations → iterator valid
- ▶ Solution: Legacy-to-STL adapters count
 - Can remove/add after iterator
 - Can replace everywhere
 - Safe also if reallocating
 - **But: Will break** when removing/adding **before** iterator

```
#ifndef NDEBUG
RooAbsArg * next() override {
    if (atEnd())
        return nullptr;
    return fSTLContainer[fIndex++];
}
#else
RooAbsArg * next() override {
    if (atEnd())
        return nullptr;
    return nextChecked();
}
#endif
```



The Challenge II

- ▶ Detect broken loops:
 - Unless NDEBUG set, legacy iterators check if they point to the same element before incrementing
 - Fail noisily if not
- ▶ Does this happen?
 - RooFit: two in hundreds of loops
 - User code: Users usually just forward-iterate

```
#ifndef NDEBUG
RooAbsArg * next() override {
    if (atEnd())
        return nullptr;
    return fSTLContainer[fIndex++];
}
#else
RooAbsArg * next() override {
    if (atEnd())
        return nullptr;
    return nextChecked();
}
#endif
```

```
#ifndef NDEBUG
template<class STLContainer>
RooAbsArg * TIteratorToSTLInterface<STLContainer>::nextChecked() {
    RooAbsArg * ret = fSTLContainer.at(fIndex);
    if (fCurrentElem != nullptr && ret != fCurrentElem) {
        throw std::logic_error("A RooCollection should not be modified while iterating. "
            "Only inserting at end is acceptable.");
    }
    fCurrentElem = ++fIndex < fSTLContainer.size() ? fSTLContainer[fIndex] : nullptr;
    return ret;
}
#endif
```

Is this the right way to notify users?



The Legacy Iterators now

- ▶ All legacy iterators work
- ▶ Slower than before
- ▶ Deprecated in Doxygen

- ▶ Flagged with

R__SUGGEST_FUNCTION*:

- Requested during user's workshop
- Flags functions/classes whose use is discouraged, but won't be fully deprecated
- <https://github.com/root-project/root/pull/3100>

```
C Compare Viewer
Local: RooAbsCollection.h
1108 Bool_t overlaps(const RooAbsCollection& otherColl) const ;
1109
1110 // \deprecated TIterator-style iteration over contained elements. Use begin() and end() or
1111 // range-based for loop instead.
1112 inline TIterator* createIterator(Bool_t dir = kIterForward) const
1113 R__SUGGEST_FUNCTION("begin(), end() and range-based for loops.") {
1114 // Create and return an iterator over the elements in this collection
1115 return new RooLinkedListIter(makeLegacyIterator(dir));
1116 }
1117
1118 // \deprecated TIterator-style iteration over contained elements. Use begin() and end() or
1119 // range-based for loop instead.
1120 RooLinkedListIter iterator(Bool_t dir = kIterForward) const
1121 R__SUGGEST_FUNCTION("begin(), end() and range-based for loops.") {
1122 return RooLinkedListIter(makeLegacyIterator(dir));
1123 }
1124
1125 // \deprecated One-time forward iterator. Use begin() and end() or
1126 // range-based for loop instead.
1127 RooFIter fwdIterator() const
1128 R__SUGGEST_FUNCTION("begin(), end() and range-based for loops.") {
1129 return RooFIter(makeLegacyIterator());
1130 }
1131
```

* Names not finalised. Opinions?



The Legacy Iterators now

```
#define R__SUGGEST_FASTER_FUNCTIONS*
```

```
root-src/roofit/roofitcore/src/RooAbsCollection.cxx:725:
```

```
21: warning: 'fwdIterator' is deprecated:
```

```
This function has faster/more secure alternatives:
```

```
begin(), end() and range-based for loops.
```

```
[-Wdeprecated-declarations]
```

```
    RooFIter iter = fwdIterator() ;
```

^

workshop

- Flags functions/classes whose use is discouraged, but won't be fully deprecated
- <https://github.com/root-project/root/pull/3100>

```
123 }
124
125 /// \deprecated One-time forward iterator. Use begin() and end() or
126 /// range-based for loop instead.
127 RooFIter fwdIterator() const
128 R__SUGGEST_FUNCTION("begin(), end() and range-based for loops.") {
129     return RooFIter(makeLegacyIterator());
130 }
131
```

```
ion& otherColl) const ;
```

```
ration over contained elements. Use begin() and end() or
```

```
ol_t dir = kIterForward) const
    and range-based for loops.") {
    over the elements in this collection
    LegacyIterator(dir));
```

```
ration over contained elements. Use begin() and end() or
```

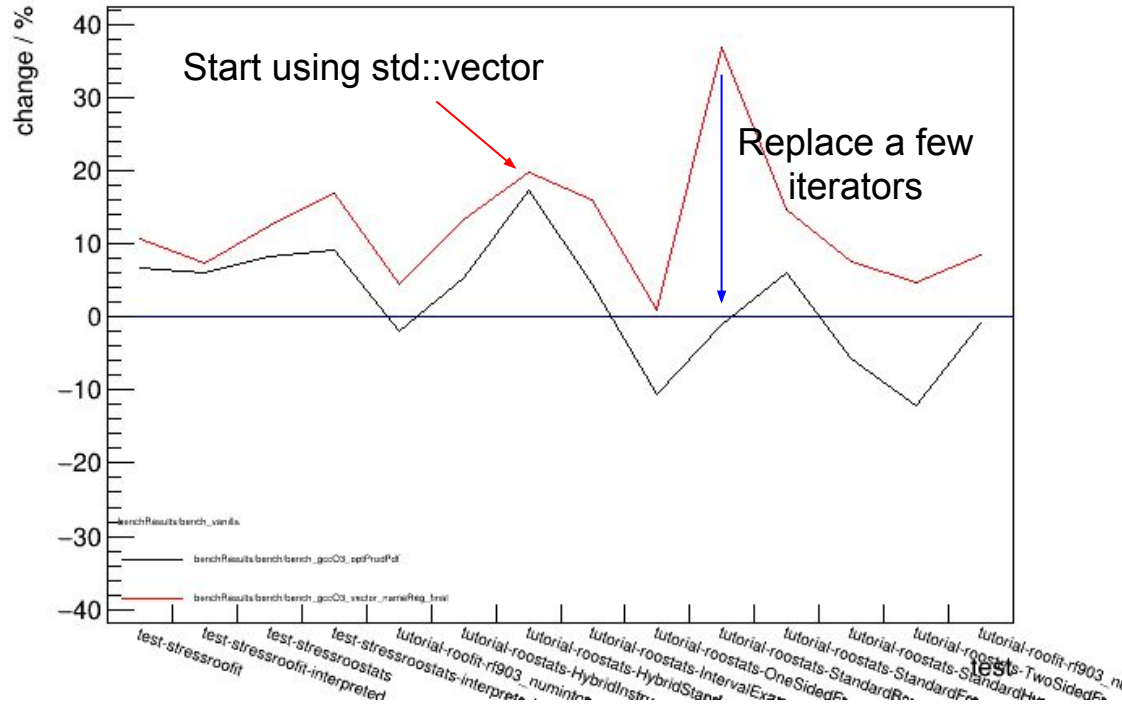
```
lir = kIterForward) const
    and range-based for loops.") {
    cyIterator(dir));
```

* Names not finalised. Opinions?



First Speed Tests

RooFit Ctests > 10s



- ▶ No standardised benchmarks for RooFit
- ▶ My solution:
 - Run all ctests 5x
 - Calculate truncated mean of best 4 runs
- ▶ **Result:**
When legacy iterators are used heavily, RooFit is ~20% slower
- ▶ Then: Repeat following cycle
 - VTunes
 - Check what's slow
 - Replace LegacyIterator by for (auto elm : collection)



Legacy Iterators are Easy to Find

Intel VTune Amplifier

Hotspots Hotspots by CPU Utilization

Analysis Configuration Collection Log Summary Bottom-up Caller/Callee Top-down Tree Platform

Grouping: Function / Call Stack

| Function / Call Stack | CPU Time | Instructions Retired | Microarchitecture Usage | Context Switch Time | Context Switch |
|---|-----------|----------------------|-------------------------|---------------------|----------------|
| RooProduct::evaluate | 811.876ms | 8,064,000,000 | 69.4% | | |
| std::make_unique<TIteratorToSTLInterface> | 775.793ms | 7,041,600,000 | 60.9% | | |
| RooAbsCollection::makeLegacyIterator | 775.793ms | 7,041,600,000 | 60.9% | | |
| RooAbsCollection::fwdIterator | 351.813ms | 2,944,800,000 | 55.0% | | |
| RooAbsCollection::fwdIterator | 308.713ms | 2,998,800,000 | 68.8% | | |
| RooAbsCollection::fwdIterator | 40.093ms | 352,800,000 | 54.6% | | |
| RooAbsCollection::fwdIterator | 31.072ms | 381,600,000 | 83.3% | | |
| RooAbsCollection::fwdIterator | 20.046ms | 100,800,000 | 32.1% | | |
| RooAbsCollection::fwdIterator | 16.037ms | 165,600,000 | 57.9% | | |
| RooAbsCollection::fwdIterator | 8.019ms | 97,200,000 | 27.4% | | |
| RooAbsArg::setValueDirty | 731.691ms | 5,205,600,000 | 57.9% | | |
| TIteratorToSTLInterface<std::vector<RooAbsReal>>::getValV | 597.380ms | 4,003,200,000 | 59.8% | | |
| RooAbsReal::getValV | 513.286ms | 3,351,600,000 | 45.9% | | |
| RooAbsReal::getVal | 353.818ms | 1,584,000,000 | 41.4% | | |
| RooAbsData::checkInited | 334.774ms | 1,771,200,000 | 43.7% | | |
| RooFilterForLinkedList::next | 326.755ms | 2,570,400,000 | 62.0% | | |
| RooRealSumPdf::evaluate | 314.727ms | 1,317,600,000 | 41.8% | | |
| TIteratorToSTLInterface<std::vector<RooAbsReal>>::traceEval | 236.547ms | 1,378,800,000 | 59.0% | | |
| RooAbsReal::traceEval | 230.533ms | 1,753,200,000 | 47.1% | | |
| std::make_unique<RooFilterForLinkedList> | 229.516ms | 1,310,400,000 | 38.7% | | |
| RooLinkedListIter::Reset | 194.449ms | 727,200,000 | 48.6% | | |
| libm_log_l9 | 191.442ms | 1,026,000,000 | 35.9% | | |
| RooAbsPdf::getValV | 184.426ms | 828,000,000 | 33.3% | | |
| TIteratorToSTLInterface<std::vector<RooAbsReal>>::getLogVal | 157.364ms | 1,202,400,000 | 52.1% | | |

CPU Time

Viewing 1 of 78 selected stack(s)

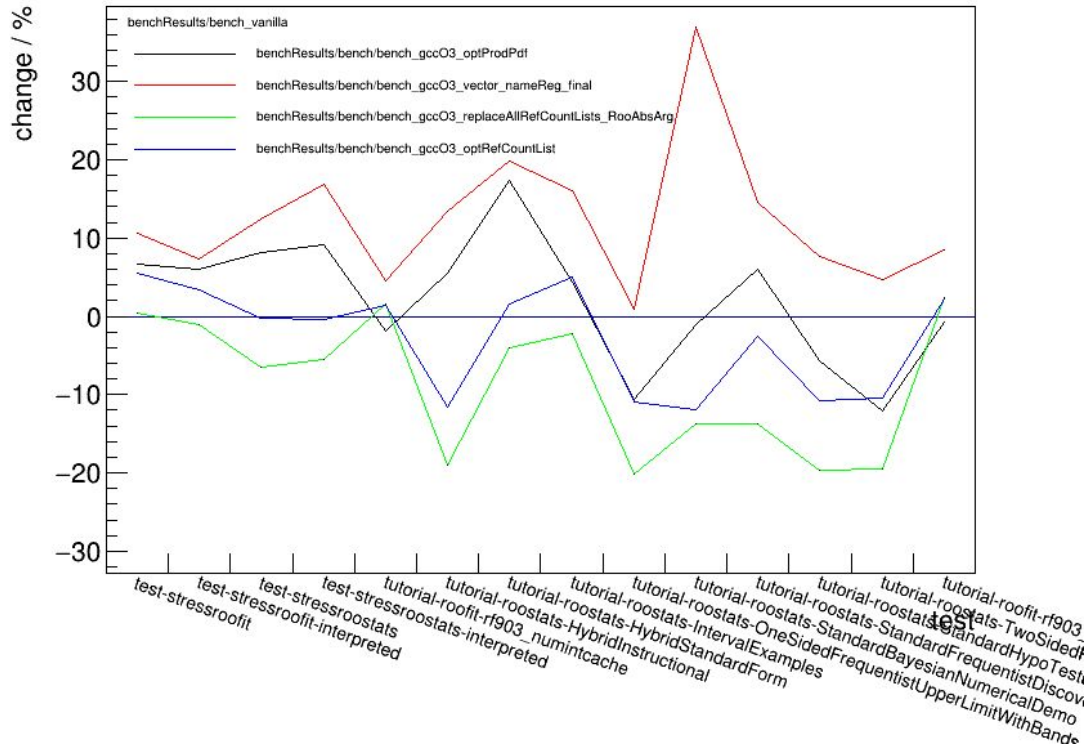
9.6% (0.074s of 0.776s)

```
libRooFitCore.so!std::make_unique<TIteratorToSTLInterface<std::vector<RooAbsReal>>>
libRooFitCore.so!RooAbsCollection::makeLegacyIterator+0x5b - RooAbsCollection.h:136
libRooFitCore.so!RooAbsCollection::fwdIterator+0x18 - RooAbsCollection.h:136
libRooFitCore.so!RooProduct::evaluate+0xac - RooProduct.cxx:375
libRooFitCore.so!RooAbsReal::traceEval+0xf - RooAbsReal.cxx:289
libRooFitCore.so!RooAbsReal::getValV+0x84 - RooAbsReal.cxx:256
libRooFitCore.so!RooAbsReal::getVal+0x2a - RooAbsReal.h:67
libRooFitCore.so!RooProduct::evaluate+0x5e - RooProduct.cxx:372
libRooFitCore.so!RooAbsReal::traceEval+0xf - RooAbsReal.cxx:289
libRooFitCore.so!RooAbsReal::getValV+0x84 - RooAbsReal.cxx:256
libRooFitCore.so!RooAbsReal::getVal+0x2a - RooAbsReal.h:67
libRooFitCore.so!RooProduct::evaluate+0x5e - RooProduct.cxx:372
libRooFitCore.so!RooAbsReal::traceEval+0xf - RooAbsReal.cxx:289
libRooFitCore.so!RooAbsReal::getValV+0x84 - RooAbsReal.cxx:256
libRooFitCore.so!RooAbsReal::getVal+0x2d - RooAbsReal.h:67
libRooFitCore.so!RooRealSumPdf::evaluate+0x96 - RooRealSumPdf.cxx:242
libRooFitCore.so!RooAbsPdf::getValV+0xeb - RooAbsPdf.cxx:287
libRooFitCore.so!RooAbsReal::getVal+0x31 - RooAbsReal.h:67
libRooFitCore.so!RooProdPdf::calculate+0x319 - RooProdPdf.cxx:549
libRooFitCore.so!RooProdPdf::evaluate+0x1e3 - RooProdPdf.cxx:493
libRooFitCore.so!RooAbsPdf::getValV+0xeb - RooAbsPdf.cxx:287
libRooFitCore.so!RooAbsReal::getVal+0x2d - RooAbsReal.h:67
libRooFitCore.so!RooAbsPdf::getLogVal+0x8 - RooAbsPdf.cxx:609
```



Speed Tests II

Roofit Ctests > 10s



- ▶ Couple of iterations of:
 - VTunes
 - Check what's slow
 - Replace LegacyIterator by for (auto elm : collection)
- ▶ RooFit faster than before
- ▶ **To do:**
 - Replace more iterators
 - Old collections only contain RooAbsArg*:

```
for (auto elm : list) {
  auto realVar = static_cast<RooRealVar*>(elm)
}
```

- ▶ New collection is templated!
 - No time to replace properly ...



Addendum: RooRefCountList

- ▶ While profiling, RooLinkedList still showed up
- ▶ Turns out to be a reference-counting list
 - Saves an element and a number
 - When iterating, ref count is not used
 - Fully represented by `std::vector<Element_t>`
`std::vector<size_t>`

```
class RooRefCountList : public RooLinkedList {
public:
    RooRefCountList() ;
    virtual ~RooRefCountList() {} ;

    virtual void Add(TObject* arg) { Add(arg,1) ; }
    virtual void Add(TObject* obj, Int_t count) ;
    virtual Bool_t Remove(TObject* obj) ;
    virtual Bool_t RemoveAll(TObject* obj) ;
    Int_t refCount(TObject* obj) const ;
};
```



RooRefCountList → RooSTLRefCountList

- ▶ RooRefCountList reimplemented as header-only RooSTLRefCountList
- ▶ + conversion functions for I/O
- ▶ Seamlessly replaces RooRefCountList
 - Should change iterators, though:
→ faster
- ▶ Internal to RooFit
→ Users shouldn't notice (just faster)

- ▶ **To do:**
 - Only slow & heavily-used instances have been replaced

```
template <class T>
class RooSTLRefCountList {
public:
    using Container_t = std::vector<T*>;
    {...}

    ///Find an item by comparing its adress.
    template<typename Obj_t>
    typename Container_t::const_iterator findByPointer(const Obj_t * item) {
        auto byPointer = [item](const T * listItem) {
            return listItem == item;
        };
        return std::find_if(_storage.begin(), _storage.end(), byPointer);
    }

    {...}

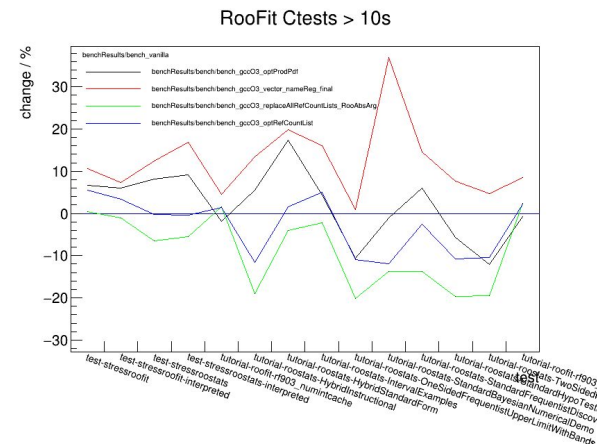
private:
    Container_t _storage;
    std::vector<std::size_t> _refCount;

    ClassDef(RooSTLRefCountList<T>,1);
};
```

@Axel: Eclipse spell-checks. Already fixed.

Summary I

- ▶ RooFit now iterates 20 - 30% faster
 - Real-world example:
My H \rightarrow bb thesis measurement
 - 11:30 min \rightarrow 9:20 min
 - 586 Mb \rightarrow 579 Mb
 - **Identical result!**
- ▶ Old iterators still work
 - Only ~25% replaced in RooFit
 - Replacing makes iteration faster
 - Slowest places presumably taken care of
 - **Caveats:**
 - Modifying collection in front of iterator not possible, any more
 - Can be detected if NDEBUG not set





- ▶ New iterators look and feel like STL
- ▶ Two branches (almost) ready:
 - <https://github.com/hageboeck/root/tree/ImproveRooAbsCollection>
 - <https://github.com/hageboeck/root/tree/ReplaceRooRefCountList>
- ▶ PRs after cleanup + a bit of doxygen here and there

```
663 void RooAbsCollection::setAttribAll(const Text_t* name  
664 {  
665     RooFIter iter= fwdIterator() ;  
666     RooAbsArg* arg ;  
667     while ((arg=iter.next())) {  
668         arg->setAttribute(name,value) ;  
669     }  
670 }
```



```
643 void RooAbsCollection::setAttribAll(const Text_t* name,  
644 {  
645     for (auto arg : _list) {  
646         arg->setAttribute(name, value);  
647     }  
648 }  
649  
650  
---
```



Backup



Speed Tests for Short RooFit Examples

RooFit Ctests < 10s

