Pythia8 — Colour reconnection optimisation

Michal Kreps

WARWICK
THE UNIVERSITY OF WARWICK

Starting point

WARWICK
THE UNIVERSITY OF WARWICK

- Couple of years back in LHCb we did Pythia8 tuning in which we used colour reconnection
  - But we have never used it because everything become very slow
- Alex Ward did some measurements and found some hotspots
- Afterwards I looked to code and found lot of small coding issues which makes code slow
- This is documentation on what and why I change to make it much faster

My measurements

WARWICK
THE UNIVERSITY OF WARWICK

- My work is based on main03 example where I change settings based on what Alex did with Peter Skands
- Colour reconnection settings are

```
ColourReconnection:mode = 1
ColourReconnection:allowDoubleJunRem = off
ColourReconnection:m0 = 0.3
ColourReconnection:allowJunctions = on
ColourReconnection:junctionCorrection = 1.20
ColourReconnection:timeDilationMode = 2
ColourReconnection:timeDilationPar = 0.18
```

- Profiling is done with 20 events, timing measurements with 5000 events
- Code before modifications runs for 2m27s,
- Profiler: 80% of Pythia8 time in colour reconnection

Shared_ptr copies

WARWICK
THE UNIVERSITY OF WARWICK

- First issue is that there are many places where copies of shared_ptr are made
- Example is

```
1170
1171       void ColourReconnection::singleJunction(ColourDipolePtr dip1,
1172   2.27   ColourDipolePtr dip2, ColourDipolePtr dip3) {
1173
1174          // Do nothing if one of the dipoles is a junction or antijunction.
1175   0.91   if (dip1->isJun || dip1->isAntiJun) return;
1176   0.68   if (dip2->isJun || dip2->isAntiJun) return;
1177   0.54   if (dip3->isJun || dip3->isAntiJun) return;
1178
1179
1180          // Check that all dipoles are active.
1181   0.67   if (!dip1->isActive || !dip2->isActive || !dip3->isActive) return;
1182
```

- Problem with this is that it invokes copy constructor and needs locking to update reference count

Shared_ptr copies

WARWICK
THE UNIVERSITY OF WARWICK

- Pass shared_ptr to colour dipole by reference wherever trivially possible
  - Test job went from 2m27s to 1m45s
- In few places it is not completely trivial, but can be improved after making function arguments const and possibly whole functions const
- Try to avoid work, which is not needed
  - checkTimeDilation function with up to 4 dipoles is good example (next slide)

Avoid unnecessary work

WARWICK
THE UNIVERSITY OF WARWICK

```
1958  0.01   } else if (dip4 == 0) {
1959  0.03     Vec4 p1 = getDipoleMomentum(dip1);
      3.15   313946 call(s) to 'Pythia8::ColourReconnection::getDipoleMomentum(std::shared_ptr<Pythia8::ColourDipole>&)' (libpythia8.so: ColourReconnection.cc, ...)
1960  0.03     Vec4 p2 = getDipoleMomentum(dip2);
      3.15   313946 call(s) to 'Pythia8::ColourReconnection::getDipoleMomentum(std::shared_ptr<Pythia8::ColourDipole>&)' (libpythia8.so: ColourReconnection.cc, ...)
1961  0.03     Vec4 p3 = getDipoleMomentum(dip3);
      3.15   313946 call(s) to 'Pythia8::ColourReconnection::getDipoleMomentum(std::shared_ptr<Pythia8::ColourDipole>&)' (libpythia8.so: ColourReconnection.cc, ...)
1962  0.04     double t1 = formationTimes[dip1->col];
      0.36   313946 call(s) to 'std::map<int, double, std::less<int>, std::allocator<std::pair<int const, double> > >::operator[](int const&)' (libpythia8.so: stl_map.h, ...)
1963  0.03     double t2 = formationTimes[dip2->col];
      0.36   313946 call(s) to 'std::map<int, double, std::less<int>, std::allocator<std::pair<int const, double> > >::operator[](int const&)' (libpythia8.so: stl_map.h, ...)
1964  0.03     double t3 = formationTimes[dip3->col];
      0.35   313946 call(s) to 'std::map<int, double, std::less<int>, std::allocator<std::pair<int const, double> > >::operator[](int const&)' (libpythia8.so: stl_map.h, ...)
1965           // Modes that require all dipoles to be causally connected.
1966  0.04     if (timeDilationMode == 1 || timeDilationMode == 2 ||
1967              timeDilationMode == 4) {
1968  0.07       if (dip1 != dip2 && !checkTimeDilation(p1, p2, t1, t2)) return false;
      1.41   313946 call(s) to 'Pythia8::ColourReconnection::checkTimeDilation(Pythia8::Vec4, Pythia8::Vec4, double, double)' (libpythia8.so: ColourReconnection.cc, ...)
1969  0.00       if (dip1 != dip3 && !checkTimeDilation(p1, p3, t1, t3)) return false;
      0.00   997 call(s) to 'Pythia8::ColourReconnection::checkTimeDilation(Pythia8::Vec4, Pythia8::Vec4, double, double)' (libpythia8.so: ColourReconnection.cc, ...)
1970  0.00       if (dip2 != dip3 && !checkTimeDilation(p2, p3, t2, t3)) return false;
      0.00   51 call(s) to 'Pythia8::ColourReconnection::checkTimeDilation(Pythia8::Vec4, Pythia8::Vec4, double, double)' (libpythia8.so: ColourReconnection.cc, ...)
1971           return true;
```

➡ Get down to 1m30s

Operations with map

- Calls like `formationTimes[dip3->col]` are costly as compiler cannot be sure whether we modify map or not
- There seems to be lot of time in singleJunction function

```
1170
1171        void ColourReconnection::singleJunction(ColourDipolePtr& dip1,
1172    3.13   ColourDipolePtr& dip2, ColourDipolePtr& dip3) {
1173
1174           // Do nothing if one of the dipoles is a junction or antijunction.
1175    1.25   if (dip1->isJun || dip1->isAntiJun) return;
1176    0.93   if (dip2->isJun || dip2->isAntiJun) return;
1177    0.74   if (dip3->isJun || dip3->isAntiJun) return;
1178
1179
1180           // Check that all dipoles are active.
1181    0.79   if (!dip1->isActive || !dip2->isActive || !dip3->isActive) return;
1182
1183           // Only allow 0-3-6, 1-4-7 or 2-5-8.
1184    2.62   if ( dip1->colReconnection % 3 != dip2->colReconnection % 3
1185    0.50     || dip1->colReconnection % 3 != dip3->colReconnection % 3) return;
1186
1187    0.41   if ( !(dip1->colReconnection != dip2->colReconnection
1188    0.10       && dip1->colReconnection != dip3->colReconnection
1189               && dip2->colReconnection != dip3->colReconnection) ) return;
```

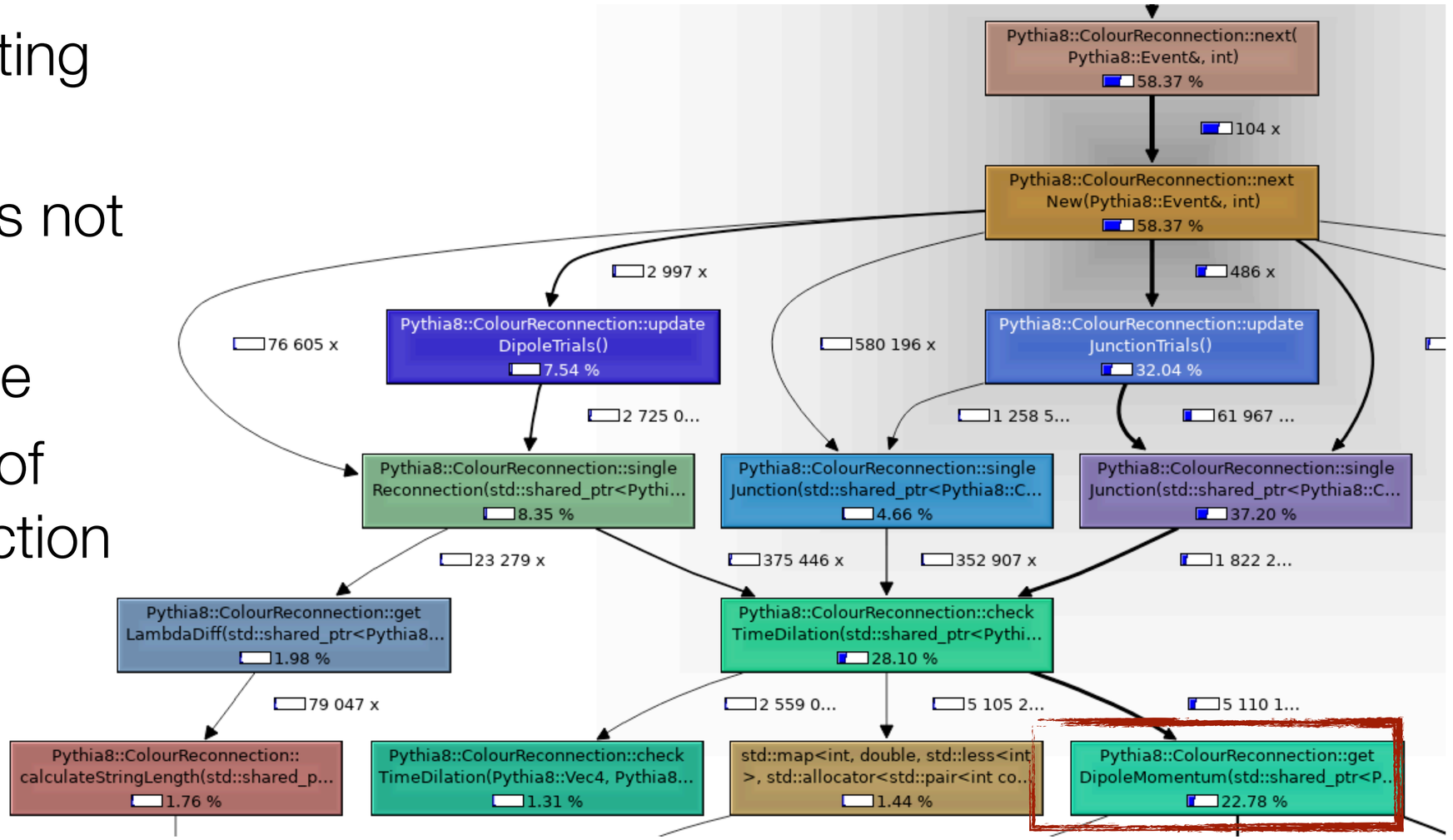
Function called in triple loop

Some checks make more sense in that loop to decrease number of if statements

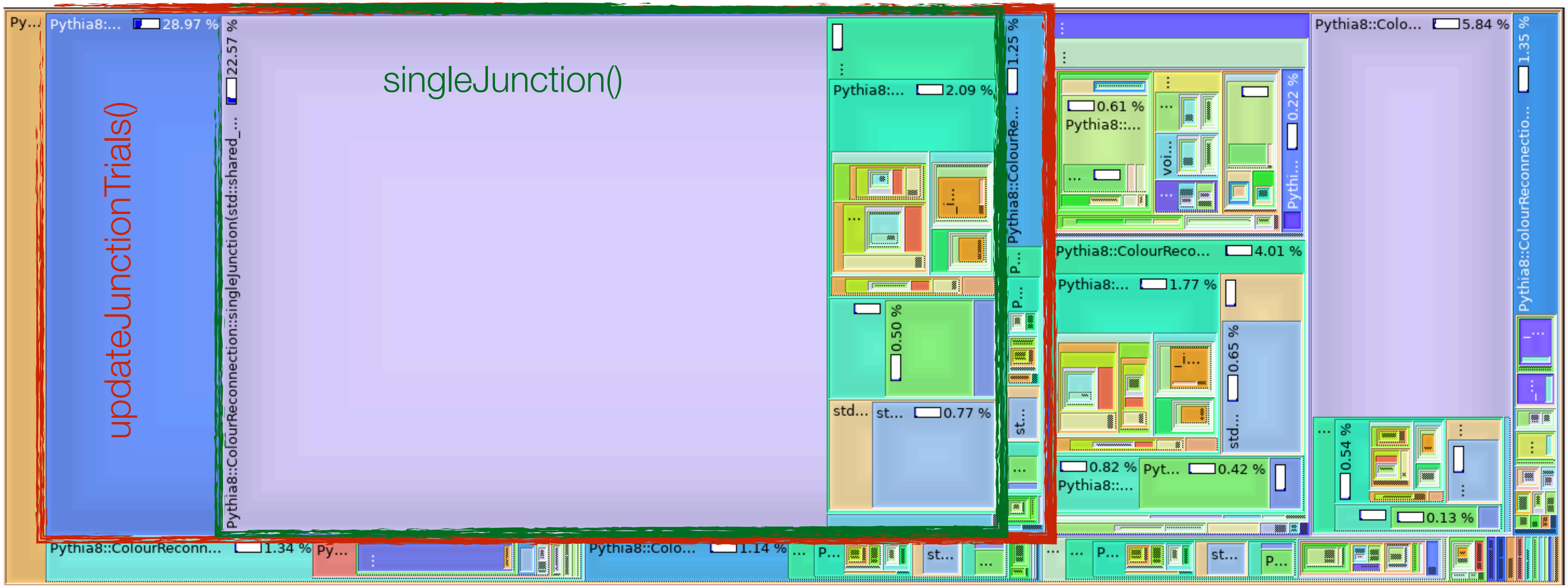
Elephant in the room

- Lot of time spent getting dipole momentum
- Most of the time does not change
- Cache where possible
- Significant decrease of time spent in the function (22% to 4%)

Change profiling statistics from 20 to 100 events



Profiling after changes



➡ Just colour reconnection

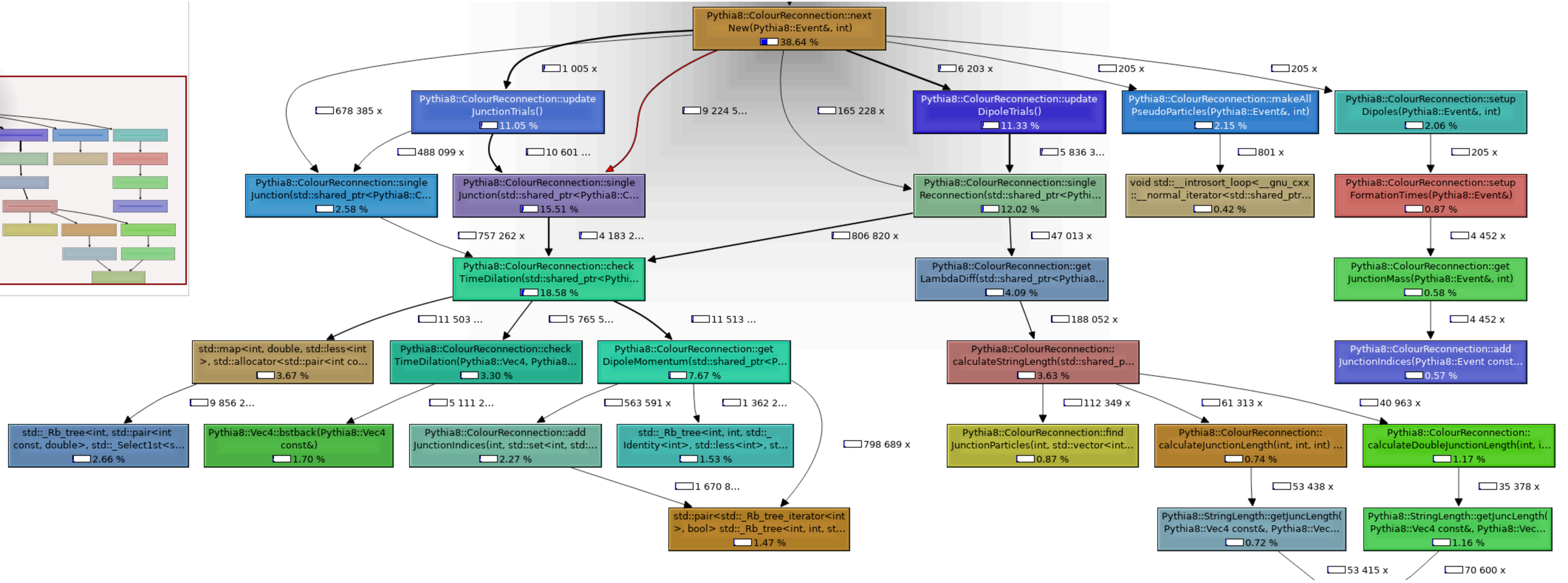
singleJunction function

WARWICK
THE UNIVERSITY OF WARWICK

- Called huge number of times and most of the times it just checks whether dipoles make sense together
- Move these checks outside to place where function is called
- With little bit of cleverness, most of the checks can be done much fewer times
- Overall effect is to decrease time of the test from 70s to 55s

Profiler after changes

WARWICK
THE UNIVERSITY OF WARWICK



Summary

WARWICK
THE UNIVERSITY OF WARWICK

- Handful of trivial changes to make code slightly more friendly to compiler
- Help memory access to pick up few things only if needed
- Couple of if statements optimised to avoid some evaluations
- Partial caching of dipole momentum
- Overall, my test goes from 2m27s to 55s
- Further improvements are likely possible (cache dipole momentum, avoid additional work etc) but no trivial place where big gain is easy
- There are probably things reevaluated many times even if they do not change