Fine-Grained Parallelism in cmsRun

Lessons learned in the attempt
Marc Paterno
13 May 2010

Project purpose

- “fine-grained” portion of effort to bring concurrency to cmsRun.
- Fine-grained means:
    - Only local modifications to code.
    - No change in *results* allowed, only change in *performance*.
- Investigate use of one of the popular “toolkits” for concurrent programming
    - Intel TBB: excellent library, but intrusive
    - OpenMP: “simple” design, not intrusive

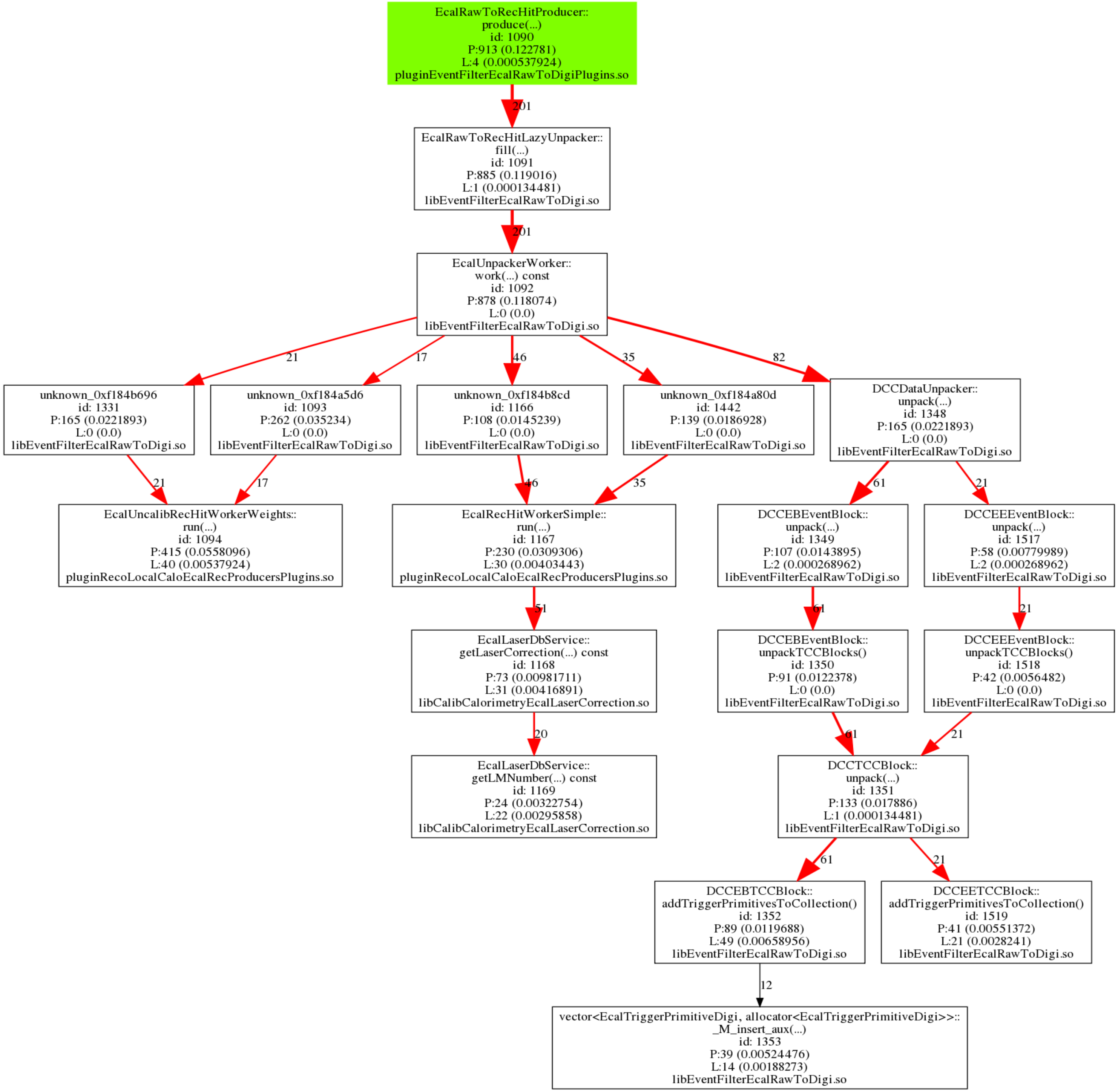
Plan of attack

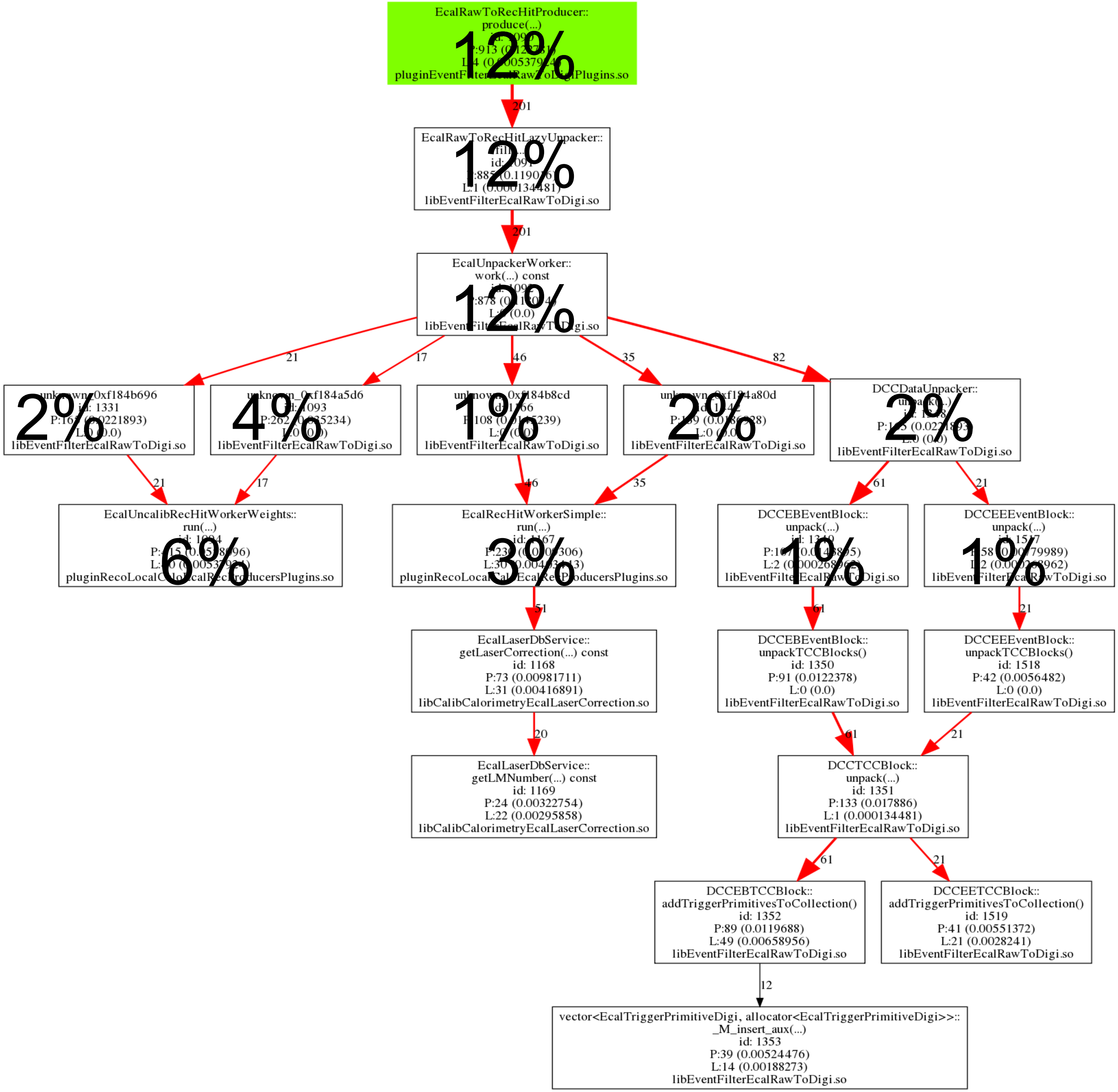
- Identify a portion of CMS code that is suitable for localized concurrency:
  - takes significant time, enough to be worth the effort – tend toward higher-level functions,
  - is not *inherently* serial,
  - has no *accidental* serialization, or can have such removed – tend toward lower-level functions,
  - deals with sufficient data to benefit from OpenMP-style concurrency (*e.g.*, parallelization of loops).
- See if application of OpenMP improves speed.

- First looked at reconstruction executable
  - ttbar simulation sample
  - CMSSW_3_1_0, arch=slc4_ia32_gcc432
- Revisited with newer executable, running HLT
  - Simulated L1 trigger skim sample
  - CMSSW_3_2_1, arch=slc4_ia32_gcc432
  - CMSSW_3_3_0, arch_slc5_amd64_gcc432

cmsRun in HLT

- Analysis of profiling data turned up one good candidate: EcalRawToRecHitProducer::produce (~12% of total program time)
- The following slide shows the (trimmed of rare path) call paths
- Investigation of these routines revealed much (accidental?) serialization – local changes could not introduce useful parallelization.


EcalRawToRecHitProducer::
produce(...)
id: 1090
P:913 (0.122781)
L:4 (0.000537924)
pluginEventFilterEcalRawToDigiPlugins.so
201
EcalRawToRecHitLazyUnpacker::
fill(...)
id: 1091
P:885 (0.119016)
L:1 (0.000134481)
libEventFilterEcalRawToDigi.so
201
EcalUnpackerWorker::
work(...) const
id: 1092
P:878 (0.118074)
L:0 (0.0)
libEventFilterEcalRawToDigi.so
21
17
46
35
82
unknown_0xf184b696
id: 1331
P:165 (0.0221893)
L:0 (0.0)
libEventFilterEcalRawToDigi.so
unknown_0xf184a5d6
id: 1093
P:262 (0.035234)
L:0 (0.0)
libEventFilterEcalRawToDigi.so
unknown_0xf184b8cd
id: 1166
P:108 (0.0145239)
L:0 (0.0)
libEventFilterEcalRawToDigi.so
unknown_0xf184a80d
id: 1442
P:139 (0.0186928)
L:0 (0.0)
libEventFilterEcalRawToDigi.so
DCCDataUnpacker::
unpack(...)
id: 1348
P:165 (0.0221893)
L:0 (0.0)
libEventFilterEcalRawToDigi.so
21
17
46
35
61
21
EcalUncalibRecHitWorkerWeights::
run(...)
id: 1094
P:415 (0.0558096)
L:40 (0.00537924)
pluginRecoLocalCaloEcalRecProducersPlugins.so
EcalRecHitWorkerSimple::
run(...)
id: 1167
P:230 (0.0309306)
L:30 (0.00403443)
pluginRecoLocalCaloEcalRecProducersPlugins.so
DCCEBEventBlock::
unpack(...)
id: 1349
P:107 (0.0143895)
L:2 (0.000268962)
libEventFilterEcalRawToDigi.so
DCCEEEventBlock::
unpack(...)
id: 1517
P:58 (0.00779989)
L:2 (0.000268962)
libEventFilterEcalRawToDigi.so
51
61
21
EcalLaserDbService::
getLaserCorrection(...) const
id: 1168
P:73 (0.00981711)
L:31 (0.00416891)
libCalibCalorimetryEcalLaserCorrection.so
DCCEBEventBlock::
unpackTCCBlocks()
id: 1350
P:91 (0.0122378)
L:0 (0.0)
libEventFilterEcalRawToDigi.so
DCCEEEventBlock::
unpackTCCBlocks()
id: 1518
P:42 (0.0056482)
L:0 (0.0)
libEventFilterEcalRawToDigi.so
20
61
21
EcalLaserDbService::
getLMNumber(...) const
id: 1169
P:24 (0.00322754)
L:22 (0.00295858)
libCalibCalorimetryEcalLaserCorrection.so
DCCTCCBlock::
unpack(...)
id: 1351
P:133 (0.017886)
L:1 (0.000134481)
libEventFilterEcalRawToDigi.so
61
21
DCCEBTCCBlock::
addTriggerPrimitivesToCollection()
id: 1352
P:89 (0.0119688)
L:49 (0.00658956)
libEventFilterEcalRawToDigi.so
DCCEETCCBlock::
addTriggerPrimitivesToCollection()
id: 1519
P:41 (0.00551372)
L:21 (0.0028241)
libEventFilterEcalRawToDigi.so
12
vector<EcalTriggerPrimitiveDigi, allocator<EcalTriggerPrimitiveDigi>>::
_M_insert_aux(...)
id: 1353
P:39 (0.00524476)
L:14 (0.00188273)
libEventFilterEcalRawToDigi.so


EcalRawToRecHitProducer::
produce(...)
12%
pluginEventFilterEcalRawToDigiPlugins.so
201
EcalRawToRecHitLazyUnpacker::
12%
id: 1091
L:1 (0.000134481)
libEventFilterEcalRawToDigi.so
201
EcalUnpackerWorker::
work(...) const
12%
libEventFilterEcalRawToDigi.so
21
17
46
35
82
2%
id: 1331
libEventFilterEcalRawToDigi.so
unknown_0xf184a5d6
4%
id: 1093
libEventFilterEcalRawToDigi.so
1%
libEventFilterEcalRawToDigi.so
2%
libEventFilterEcalRawToDigi.so
DCCDataUnpacker::
2%
libEventFilterEcalRawToDigi.so
21
17
46
35
61
21
EcalUncalibRecHitWorkerWeights::
run(...)
6%
EcalRecHitWorkerSimple::
run(...)
3%
pluginRecoLocalCaloEcalRecProducersPlugins.so
DCCEBEventBlock::
unpack(...)
1%
DCCEEEventBlock::
unpack(...)
1%
51
61
21
EcalLaserDbService::
getLaserCorrection(...) const
id: 1168
P:73 (0.00981711)
L:31 (0.00416891)
libCalibCalorimetryEcalLaserCorrection.so
DCCEBEventBlock::
unpackTCCBlocks()
id: 1350
P:91 (0.0122378)
L:0 (0.0)
libEventFilterEcalRawToDigi.so
DCCEEEventBlock::
unpackTCCBlocks()
id: 1518
P:42 (0.0056482)
L:0 (0.0)
libEventFilterEcalRawToDigi.so
20
61
21
EcalLaserDbService::
getLMNumber(...) const
id: 1169
P:24 (0.00322754)
L:22 (0.00295858)
libCalibCalorimetryEcalLaserCorrection.so
DCCTCCBlock::
unpack(...)
id: 1351
P:133 (0.017886)
L:1 (0.000134481)
libEventFilterEcalRawToDigi.so
61
21
DCCEBTCCBlock::
addTriggerPrimitivesToCollection()
id: 1352
P:89 (0.0119688)
L:49 (0.00658956)
libEventFilterEcalRawToDigi.so
DCCEETCCBlock::
addTriggerPrimitivesToCollection()
id: 1519
P:41 (0.00551372)
L:21 (0.0028241)
libEventFilterEcalRawToDigi.so
12
vector<EcalTriggerPrimitiveDigi, allocator<EcalTriggerPrimitiveDigi>>::
_M_insert_aux(...)
id: 1353
P:39 (0.00524476)
L:14 (0.00188273)
libEventFilterEcalRawToDigi.so

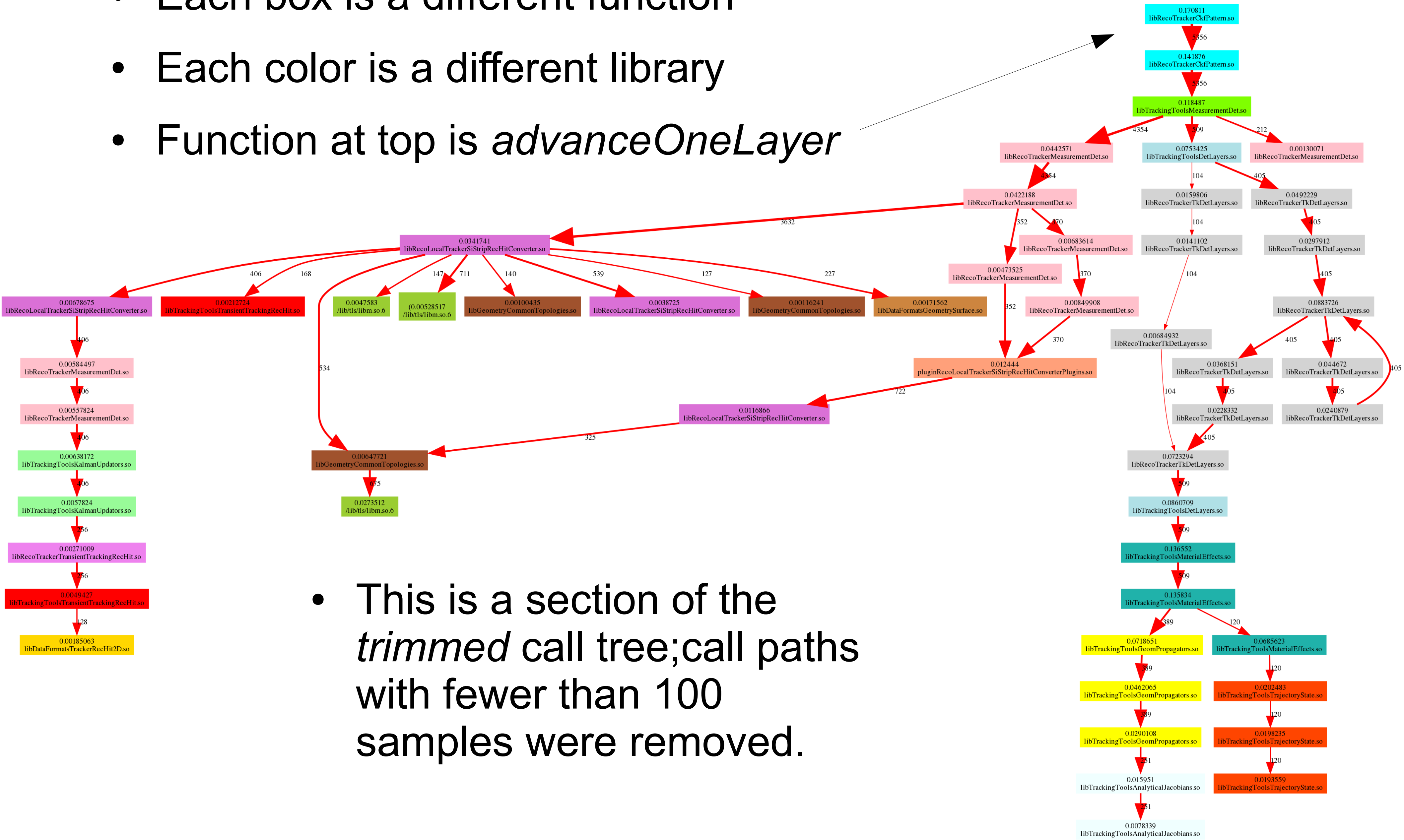
Hard-to-parallelize code structure

- Common usage of OO techniques makes for code that is not easy to parallelize
  - OO techniques *encapsulate* state for ease of understanding the code...
  - but encapsulated state, when also *shared*, prevents parallelization.
- Much of this sharing is *accidental* not *essential*.
- Maybe we need to learn from the *functional* programming community
  - pass state *to* algorithms when we want parallelism.

cmsRun in reconstruction

- Output functions use considerable time but are not good candidates for *local* multithreading
  - parallel-capable i/o formats would be interesting
- As anticipated, tracking takes the most time.
- GroupedCkfTrajectoryBuilder::advanceOneLayer, and functions it calls, take 17% of program time.
- Analysis of this code also shows great complexity.

- Each box is a different function
- Each color is a different library
- Function at top is *advanceOneLayer*


0.170811
libRecoTrackerCkfPattern.so
5356
0.141876
libRecoTrackerCkfPattern.so
5356
0.118487
libTrackingToolsMeasurementDet.so
4354
509
212
0.0442571
libRecoTrackerMeasurementDet.so
0.0753425
libTrackingToolsDetLayers.so
0.00130071
libRecoTrackerMeasurementDet.so
4354
104
405
0.0422188
libRecoTrackerMeasurementDet.so
0.0159806
libRecoTrackerTkDetLayers.so
0.0492229
libRecoTrackerTkDetLayers.so
3632
352
370
104
405
0.0341741
libRecoLocalTrackerSiStripRecHitConverter.so
0.00683614
libRecoTrackerMeasurementDet.so
0.0141102
libRecoTrackerTkDetLayers.so
0.0297912
libRecoTrackerTkDetLayers.so
0.00473525
libRecoTrackerMeasurementDet.so
406
168
147
711
140
539
127
227
370
104
405
0.00678675
libRecoLocalTrackerSiStripRecHitConverter.so
0.00212724
libTrackingToolsTransientTrackingRecHit.so
0.0047583
/lib/tls/libm.so.6
(0.00528517
/lib/tls/libm.so.6
0.00100435
libGeometryCommonTopologies.so
0.0038725
libRecoLocalTrackerSiStripRecHitConverter.so
0.00116241
libGeometryCommonTopologies.so
0.00171562
libDataFormatsGeometrySurface.so
352
0.00849908
libRecoTrackerMeasurementDet.so
0.0883726
libRecoTrackerTkDetLayers.so
406
534
370
0.00684932
libRecoTrackerTkDetLayers.so
405
405
405
0.00584497
libRecoTrackerMeasurementDet.so
0.012444
pluginRecoLocalTrackerSiStripRecHitConverterPlugins.so
0.0368151
libRecoTrackerTkDetLayers.so
0.044672
libRecoTrackerTkDetLayers.so
406
722
104
405
405
0.00557824
libRecoTrackerMeasurementDet.so
0.0116866
libRecoLocalTrackerSiStripRecHitConverter.so
0.0228332
libRecoTrackerTkDetLayers.so
0.0240879
libRecoTrackerTkDetLayers.so
406
325
405
0.00638172
libTrackingToolsKalmanUpdators.so
0.00647721
libGeometryCommonTopologies.so
0.0723294
libRecoTrackerTkDetLayers.so
406
675
509
0.0057824
libTrackingToolsKalmanUpdators.so
0.0273512
/lib/tls/libm.so.6
0.0860709
libTrackingToolsDetLayers.so
256
509
0.00271009
libRecoTrackerTransientTrackingRecHit.so
0.136552
libTrackingToolsMaterialEffects.so
256
509
0.0049427
libTrackingToolsTransientTrackingRecHit.so
0.135834
libTrackingToolsMaterialEffects.so
128
389
120
0.00185063
libDataFormatsTrackerRecHit2D.so
0.0718651
libTrackingToolsGeomPropagators.so
0.0685623
libTrackingToolsMaterialEffects.so
389
120
0.0462065
libTrackingToolsGeomPropagators.so
0.0202483
libTrackingToolsTrajectoryState.so
389
120
0.0290108
libTrackingToolsGeomPropagators.so
0.0198235
libTrackingToolsTrajectoryState.so
251
120
0.015951
libTrackingToolsAnalyticalJacobians.so
0.0193559
libTrackingToolsTrajectoryState.so
251
0.0078339
libTrackingToolsAnalyticalJacobians.so

- This is a section of the *trimmed* call tree;call paths with fewer than 100 samples were removed.

Low-level concurrency for the future

- Accidents of current code prevent concurrency.
  - We need to “think parallel” up front.
  - We need to investigate parallel algorithms and data structures for higher-level tasks.
  - We need to devise and enforce easy-to-follow rules for making modules thread-safe.
- We need to understand how to interact with non-thread-safe utilities:
  - limit exposure in our own code
  - provide thread-safe patterns of use
  - work toward achieving thread-safety in utilities

What can we learn from others?

- Functional programming community
  - encapsulate higher-order functions
  - pass algorithm state to algorithms (reduce accidental sharing, make essential sharing explicit)
  - allows for optimizations that can be *proven* correct
- “Parallel” programming languages (Chapel, F-Script, Fortran 2008)
  - Use *whole aggregate* transformations & algorithms
  - Allow for libraries to provide means of parallelization

How do we do this in C++?

- Common wisdom: get it right first, then make it fast
  - But we have learned we can't afford to make it too slow first – must think parallel *early*
- Maybe we haven't done *enough* template programming – abstractions at the right level (per Stepanov's *Elements of Programming*)
- New C++ has valuable features
  - local (lambda) functions
  - better metaprogramming support

Thanks.

Trivial OpenMP example

```
#include <omp.h>
#include <iostream>
int main () {
  int th_id, nthreads;
#pragma omp parallel private(th_id)
  {
    th_id = omp_get_thread_num();
    std::cout << "Hello World from thread" << th_id << std::endl;
#pragma omp barrier
    if ( th_id == 0 ) {
      nthreads = omp_get_num_threads();
      std::cout << "There are " << nthreads << " threads" << std::endl;
    }
  }
}
```

- g++-mp-4.3 -o hello_mp -fopenmp hello_mp.cc