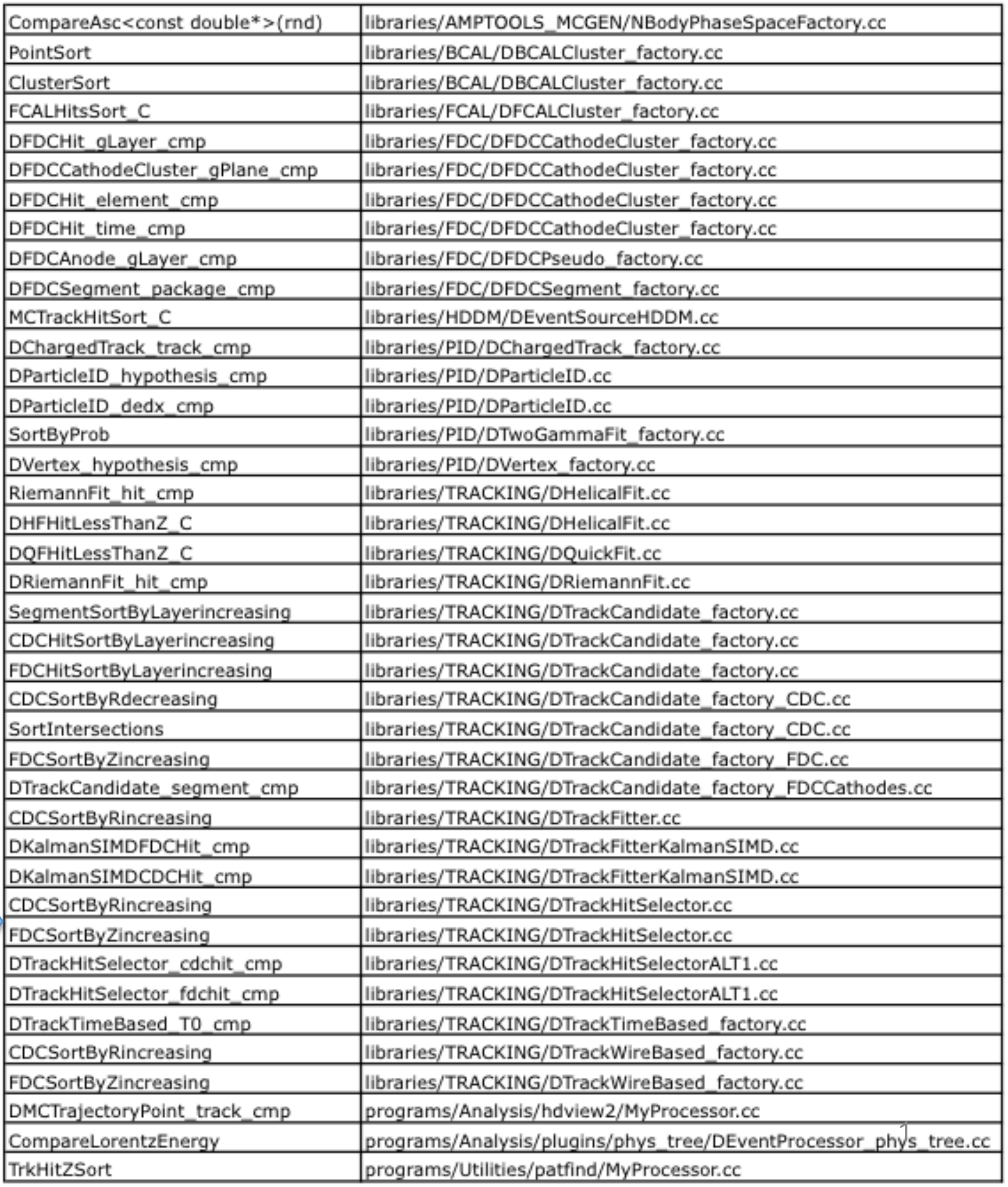
Checking *sort* comparison functions in Hall-D code

David Lawrence
April, 6, 2011

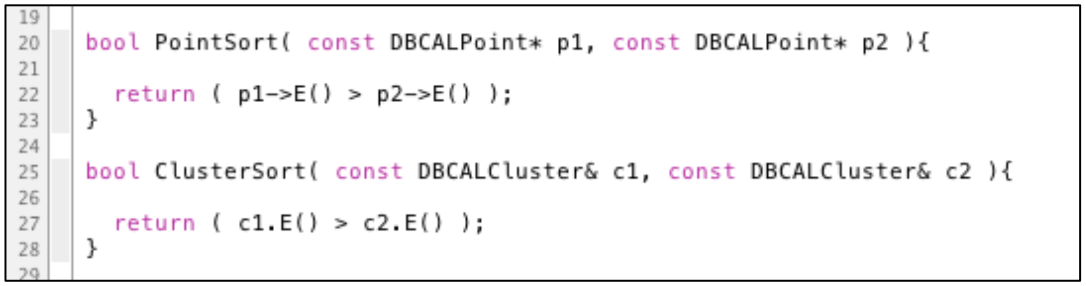
List of custom comparison functions used by us for STL sort



CompareAsc<const double*>(rnd)	libraries/AMPTOOLS_MCGEN/NBodyPhaseSpaceFactory.cc
PointSort	libraries/BCAL/DBCALCluster_factory.cc
ClusterSort	libraries/BCAL/DBCALCluster_factory.cc
FCALHitsSort_C	libraries/FCAL/DFCALCluster_factory.cc
DFDCHit_gLayer_cmp	libraries/FDC/DFDCCathodeCluster_factory.cc
DFDCCathodeCluster_gPlane_cmp	libraries/FDC/DFDCCathodeCluster_factory.cc
DFDCHit_element_cmp	libraries/FDC/DFDCCathodeCluster_factory.cc
DFDCHit_time_cmp	libraries/FDC/DFDCCathodeCluster_factory.cc
DFDCAnode_gLayer_cmp	libraries/FDC/DFDCPseudo_factory.cc
DFDCSegment_package_cmp	libraries/FDC/DFDCSegment_factory.cc
MCTrackHitSort_C	libraries/HDDM/DEventSourceHDDM.cc
DChargedTrack_track_cmp	libraries/PID/DChargedTrack_factory.cc
DParticleID_hypothesis_cmp	libraries/PID/DParticleID.cc
DParticleID_dedx_cmp	libraries/PID/DParticleID.cc
SortByProb	libraries/PID/DTwoGammaFit_factory.cc
DVertex_hypothesis_cmp	libraries/PID/DVertex_factory.cc
RiemannFit_hit_cmp	libraries/TRACKING/DHelicalFit.cc
DHFHitLessThanZ_C	libraries/TRACKING/DHelicalFit.cc
DQFHitLessThanZ_C	libraries/TRACKING/DQuickFit.cc
DRiemannFit_hit_cmp	libraries/TRACKING/DRiemannFit.cc
SegmentSortByLayerincreasing	libraries/TRACKING/DTrackCandidate_factory.cc
CDCHitSortByLayerincreasing	libraries/TRACKING/DTrackCandidate_factory.cc
FDCHitSortByLayerincreasing	libraries/TRACKING/DTrackCandidate_factory.cc
CDCSortByRdecreasing	libraries/TRACKING/DTrackCandidate_factory_CDC.cc
SortIntersections	libraries/TRACKING/DTrackCandidate_factory_CDC.cc
FDCSortByZincreasing	libraries/TRACKING/DTrackCandidate_factory_FDC.cc
DTrackCandidate_segment_cmp	libraries/TRACKING/DTrackCandidate_factory_FDCCathodes.cc
CDCSortByRincreasing	libraries/TRACKING/DTrackFitter.cc
DKalmanSIMDFDCHit_cmp	libraries/TRACKING/DTrackFitterKalmanSIMD.cc
DKalmanSIMDCDCHit_cmp	libraries/TRACKING/DTrackFitterKalmanSIMD.cc
CDCSortByRincreasing	libraries/TRACKING/DTrackHitSelector.cc
FDCSortByZincreasing	libraries/TRACKING/DTrackHitSelector.cc
DTrackHitSelector_cdchit_cmp	libraries/TRACKING/DTrackHitSelectorALT1.cc
DTrackHitSelector_fdchit_cmp	libraries/TRACKING/DTrackHitSelectorALT1.cc
DTrackTimeBased_T0_cmp	libraries/TRACKING/DTrackTimeBased_factory.cc
CDCSortByRincreasing	libraries/TRACKING/DTrackWireBased_factory.cc
FDCSortByZincreasing	libraries/TRACKING/DTrackWireBased_factory.cc
DMCTrajectoryPoint_track_cmp	programs/Analysis/hdview2/MyProcessor.cc
CompareLorentzEnergy	programs/Analysis/plugins/phys_tree/DEventProcessor_phys_tree.cc
TrkHitZSort	programs/Utilities/patfind/MyProcessor.cc

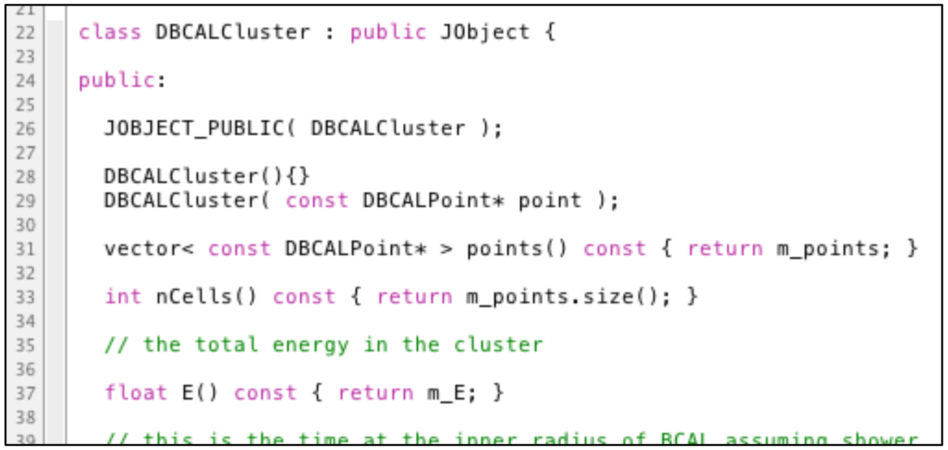
Checking for the FPU/sort bug

From *DBCALCluster_factory.cc*



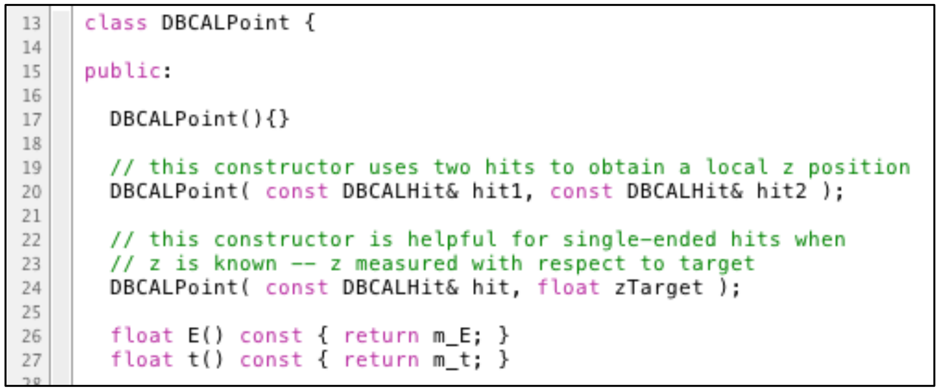
```
bool PointSort( const DBCALPoint* p1, const DBCALPoint* p2 ){

  return ( p1->E() > p2->E() );
}

bool ClusterSort( const DBCALCluster& c1, const DBCALCluster& c2 ){

  return ( c1.E() > c2.E() );
}
```

From *DBCALCluster.h*



```
class DBCALCluster : public JObject {

public:

  JOBJECT_PUBLIC( DBCALCluster );

  DBCALCluster(){}
  DBCALCluster( const DBCALPoint* point );

  vector< const DBCALPoint* > points() const { return m_points; }

  int nCells() const { return m_points.size(); }

  // the total energy in the cluster

  float E() const { return m_E; }

  // this is the time at the inner radius of BCAL assuming shower
```

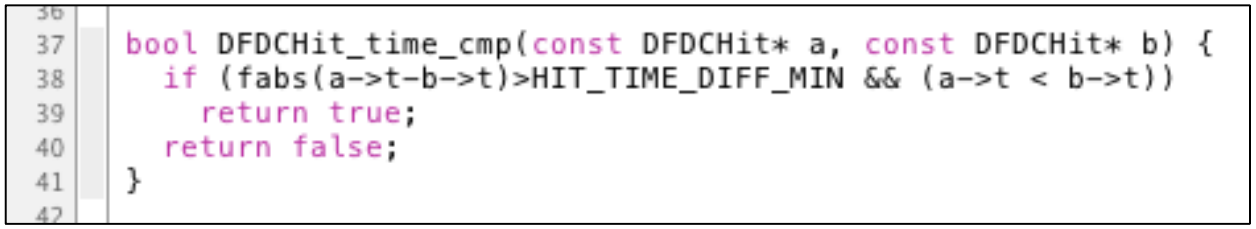
From *DBCALPoint.h*



```
class DBCALPoint {

public:

  DBCALPoint(){}

  // this constructor uses two hits to obtain a local z position
  DBCALPoint( const DBCALHit& hit1, const DBCALHit& hit2 );

  // this constructor is helpful for single-ended hits when
  // z is known -- z measured with respect to target
  DBCALPoint( const DBCALHit& hit, float zTarget );

  float E() const { return m_E; }
  float t() const { return m_t; }
```

Potential problem?

From *DFDCCathodeCluster_factory.cc*



```
bool DFDCHit_time_cmp(const DFDCHit* a, const DFDCHit* b) {
  if (fabs(a->t-b->t)>HIT_TIME_DIFF_MIN && (a->t < b->t))
    return true;
  return false;
}
```

This code does a floating point calculation of a value that is used for comparison.

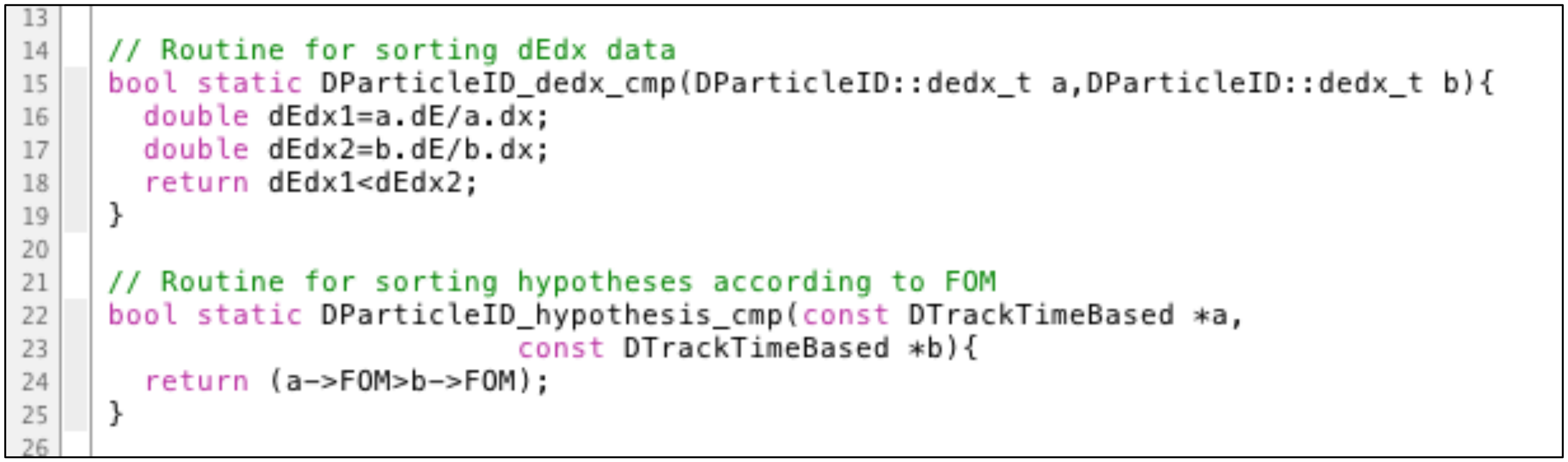
The operation is subtraction though on two values that were pre-calculated.

The pre-calculated values are 64bit so the subtraction should not populate any higher precision bits in the 80bit FPU.

This does not appear to be susceptible to the FPU/sort bug.

Another one …

From *DParticleID.cc*



```
// Routine for sorting dEdx data
bool static DParticleID_dedx_cmp(DParticleID::dedx_t a,DParticleID::dedx_t b){
  double dEdx1=a.dE/a.dx;
  double dEdx2=b.dE/b.dx;
  return dEdx1<dEdx2;
}

// Routine for sorting hypotheses according to FOM
bool static DParticleID_hypothesis_cmp(const DTrackTimeBased *a,
                                       const DTrackTimeBased *b){
  return (a->FOM>b->FOM);
}
```

The first sort algorithm here calculates values used to compare the objects using the FPU.

If object “a” and object “b” are the same, then the calculation of dEdx2 could be left at 80bit precision while dEdx1 is masked down to 64bits causing the comparison to be true when it shouldn’t

This is definitely susceptible to the FPU/sort bug!

Summary

- About 40 of our ~300 sort calls use a custom comparison algorithm
- All of these have been checked and one has been found to be susceptible to the FPU/sort bug
- Several others (that use methods) would be potentially susceptible if the internal object structure were changed