Digitization and MT

Software TIM in Glasgow

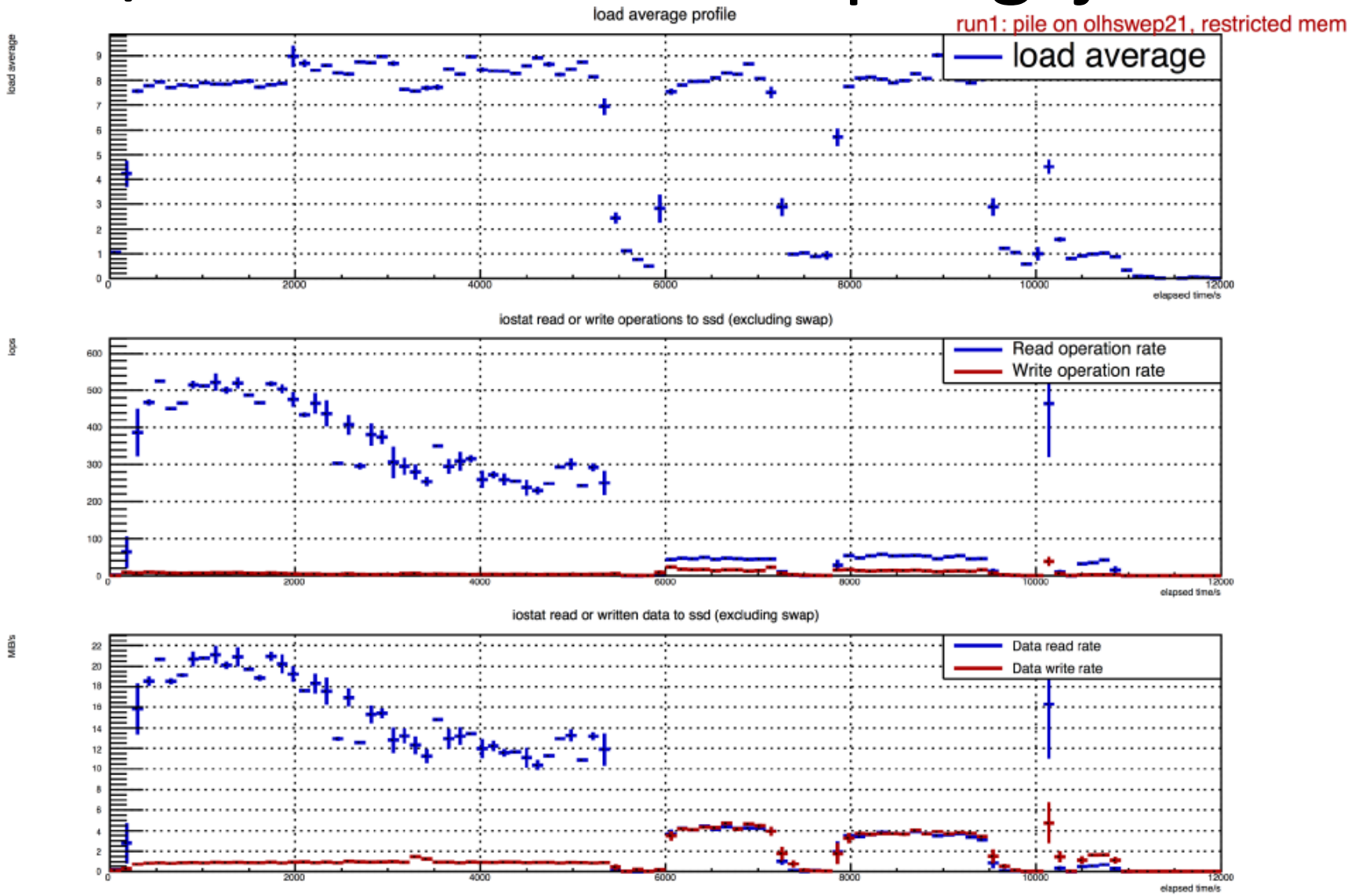
John Chapman (Cambridge)

07 June 2016

Two approaches

- **Two approaches to Pile-Up Digitization currently available in ATLAS:**
  - **Algorithm approach prioritizes less I/O at the expense of higher memory (not feasible at higher mu values).**
  - **PileUpTools approach prioritizes lower memory at the expense of more I/O (current production default and nemesis of grid-sysadmins worldwide).**

I/O load from Pile Up Digi jobs



Slide from David Smith

Core Digitization Code

- **Most code is common to the two approaches. The main differences are when background event hit collection are read-in and dropped from the job. Will come back to this.**
- **First we should look at the core pile-up digi code.**
- **Digitization uses a similar “few Algs, many Tools” design, similar to simulation.**
- **Digitization uses its own Event Loop Manger: PileUpEventLoopMgr (inherits from AthenaEventLoopMgr).**

PileUpEventLoopMgr

- **In addition to the standard stuff, PileUpEventLoopMgr is responsible for creating the PileUpEventInfo object (pre R21) or the xAOD::EventInfo object (R21 onwards). This contains all the information about which background events are to be used with the current signal event.**
- **Background event rules:**
  - **Avoid re-using the same background events for a given signal event.**
  - **Avoid re-using high pT events within a dataset.**

Bank of Bkg Events (I)

Bank of Bkg Events (II)

- **In order to ensure randomisation of background digitization jobs have to have a large enough a bank of background events. E.g. for <mu>=40 then the bank size is 1797 events. (Tuned to be as small as possible while still containing sufficient events to cover variations in mu.)**
- **Each entry in this bank has an input stream object and an instance of StoreGateSvc. (*Can we drop the INFO level messages from StoreGateSvc initialize and finalize please…*)**

Bank of Bkg Events (III)

- **In the Algorithm approach, each time a background stream is picked there is a 1/150 (tunable) chance that the event it contains will be dropped and replaced with a new one.**
- **Not clear how we would deal with multiple events in flight in this case.**
- **AthenaMP has a separate bank of events per process (effectively).**

Bank of Bkg Events (IV)

- **In the PileUpTools approach each time a background stream is used the event it contains will be dropped after being used.**
- **PileUpToolsAlg::execute()**
  - **Call IPileUpTool::prepareEvent() for all PileUpTools.**
  - **loops over the bunch-crossings for each signal event.**
    - **All the background events associated with a given bunch-crossing are passed to all interested PileUpTools for processing.**
    - **StoreGateSvc::clearStore() is called for all StoreGateSvc instances used in the current bunch-crossing.**
  - **Call IPileUpTool::mergeEvent() for all PileUpTools.**
- **Seems unlikely that these StoreGateSvc instances can be shared between threads in an MT environment with multiple events in flight.**

Threads For Sub-detector Tools?

- **Digitization of each sub-detector is orthogonal.**
- **In fact, Digitization of each module of each sub-detector is orthogonal.**
- **Large potential for sub-event parallelism, at least at the sub-detector level in this loop over bunch-crossings and possibly at the module level.**
- **Different level to what is being discussed for reco though?**

Other things to watch out for

- **PileUpTools are currently public tools (mostly easy enough to make them private).**
- **PileUpTools cache information during this loop over bunch-crossings. Not a problem if they are private tools?**

IBeamIntensity Services

- **Provides the relative beam intensity as a function of the bunch crossing.**

```
#include "GaudiKernel/IService.h"
class IBeamIntensity : virtual public IService {
public:
  ///a scale factor (average value 1.0) for the beam intensity at a given
  ///xing. Note how the xing is a signed int, relative to the t0 xing
  virtual float normFactor(int bunchXing) const = 0;

  ///the largest element in the beam intensity pattern. Required by the
  ///BkgStreamsCaches to ensure the background caches are large enough.
  virtual float largestElementInPattern() const = 0;

  ///randomly select in which bunch the current t0 is wrto the beam intensity
  /// distribution. This should be done proportionally to the distribution...
  virtual void selectT0() =0;

  ///return the bunch crossing selected to be the current t0 bunch crossing
  virtual unsigned int getCurrentT0BunchCrossing() const =0;

  ///return the length of the beam pattern
  virtual unsigned int getBeamPatternLength() const =0;

  static const InterfaceID& interfaceID() {
    static const InterfaceID IID( "IBeamIntensity", 1, 0 );
    return IID;
  }
};
#endif
```

Changes internal state of Svc. Called once per signal event.

IBeamLuminosity Services

- **Allows the beam luminosity to be scaled as a function of the run and lumiblock.**

```
#include "GaudiKernel/IService.h"
class IBeamLuminosity : virtual public IService {
public:
  ///a scale-down factor (between 0 and 1) for the beam luminosity at a given
  ///run and lumiblock number
  virtual float scaleFactor(unsigned int run, unsigned int lumi, bool & updated) = 0;

  static const InterfaceID& interfaceID() {
    static const InterfaceID IID( "IBeamLuminosity", 1, 0 );
    return IID;
  }
};
```

Changes internal state of Svc (unnecessarily). Called once per signal event.

Other Approaches

- **Fast Digitization (ID only so far). (CPU and I/O reduction at the expense of accuracy?)**
  - **Initially only deals with in-time pile-up.**
- **On the fly-pile-up generation in the Fast Chain. (I/O reduction at the expense of CPU)**
  - **Only feasible for in-time pile-up.**
- **Calo only out-of-time pile-up. (reduced I/O)**
- **Pre-mixing background events. (reduced I/O? (TBD) at the expense of decreased randomness)**

HepMcParticleLink Migration

https://its.cern.ch/jira/browse/ATLASSIM-2430

(Olivier Arnaez)

- Target Release: 20.20.4
- Aim
  - Extend the HepMcParticleLink class to allow it to point to multiple McEventCollections.
- Impact on sub-system code:
  - SimHits now have constructors which take HepMcParticleLinks as arguments, rather than just using the barcode (which is ambiguous in the case of multiple GenEvents/McEventCollections).
  - Changes to how HepMcParticleLinks are created in sub-detector digitization packages. Prefer to use copy-constructors rather than constructor taking eventId and barcode.
  - Not backwards compatible.
    - In this case code will compile against older releases, but behaviour may be altered.

xAOD::EventInfo Migration

https://its.cern.ch/jira/browse/ATLASSIM-2122

(Iain, Attila, JC)

- Target Release: 21.X.Y? 22?
- Aim
  - Replace all usage of PileUpEventInfo with xAOD::EventInfo.
- Status
  - PileUpTools now all use xAOD::EventInfo.
  - Core Digitization code reads in EventInfo objects and builds a PileUpEventInfo object which is then converted into an xAOD::EventInfo object for use elsewhere.
- Next steps
  - Tweak TP Converters to convert persistent EventInfo objects into transient xAOD::EventInfo objects.
  - Migrate core digitization code to use xAOD::EventInfo internally (done, but untested).
  - Migrate reconstruction, simulation and generation.

Code Clean-up

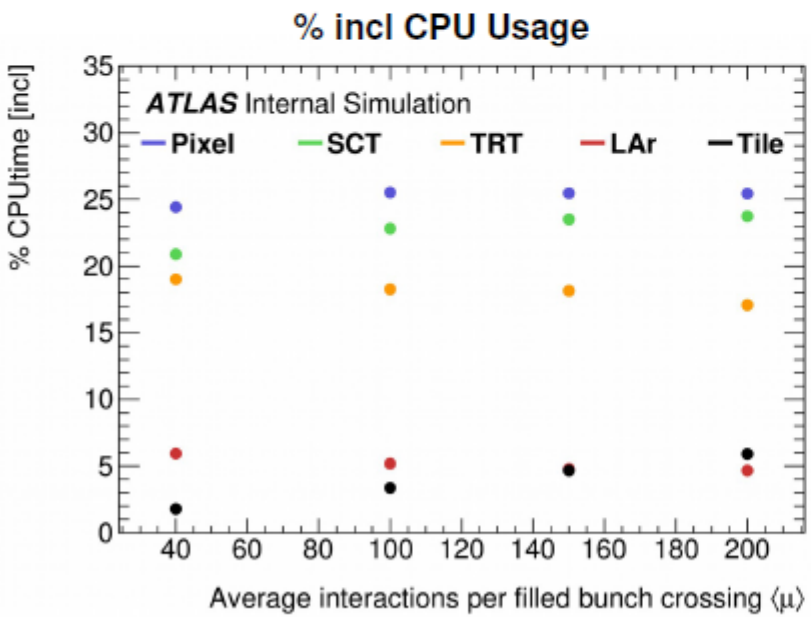
- **Core digi effort to go through PileUpTools to try to clean up old syntax/fix warnings etc.**
  - **Good progress.**
  - **Hampered by sub-system developers committing junk to the trunk.**
    - **Balance to be struck between lack of manpower and lack of understanding of workflows.**
    - **Issues with ownership of packages in some sub-systems.**
- **Clean-up of code using things like short-lived heap allocations to pick targets.**
  - **More interest from sub-system developers in fixing these.**

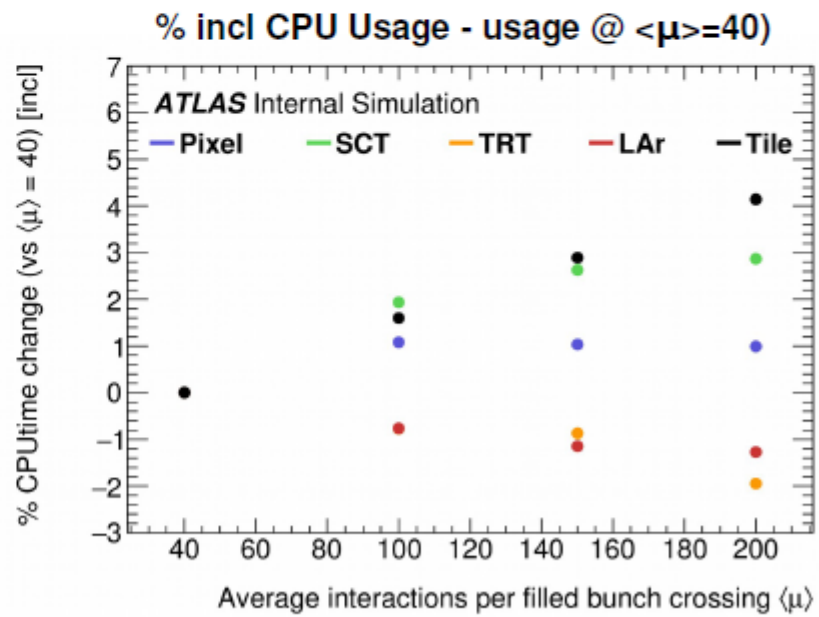
Profiling Digitization

- Next few slides show profiling of Digitization done by Hass Abouzeid, previously shown in the simulation group meeting: (https://indico.cern.ch/event/512371/)

Sub-system Ranking vs <μ>

- Ranking of sub-system CPU time doesn't change vs <μ>
- Usage vs pile-up does change
  - TRT %-time decreases slightly
  - Pixel %-time ~ steady (w/ jump from μ= 40 → 100)
  - SCT monotonically increases
  - LAr hovers around 5%
  - Tile **very** badly behaved: ~ 2% → 5%





PixelBarrelChargeTool::charge(...) /1

Usage

- Incl. ~ 14%
- Self ~ 3%

Method contains a nested for loop

- Small changes can have large effects (!!)

```
for(int istep =0; istep < nsteps; istep++) {
  double xEta1 = xEta +  stepEta * (istep + 0.5);
  double xPhi1 = xPhi +  stepPhi * (istep + 0.5);
  double depD  = xDep +  stepDep * (istep + 0.5);

  // Distance between charge and readout side.  p_design->readoutSide() is
  // +1 if readout side is in +ve depth axis direction and visa-versa.
  double spess = 0.5 * sensorThickness - Module.design().readoutSide() * depD;
  if (spess<0) spess=0;

  for(int i=0 ; i<ncharges ; i++) {
    // diffusion sigma
    double rdif=this->m_diffusionConstant*sqrt(spess*coLorentz/0.3);
    // position at the surface
    double xPhiD=xPhi1+spess*tanLorentz+rdif*CLHEP::RandGaussZiggurat::shoot(m_rndmEngine);
    double xEtaD=xEta1+rdif*CLHEP::RandGaussZiggurat::shoot(m_rndmEngine);

    // Get the charge position in Reconstruction local coordinates.
    SiLocalPosition chargePos = Module.hitLocalToLocal(xEtaD, xPhiD);

    // The parametrization of the sensor efficiency (if needed)
    double ed=e1*this->electronHolePairsPerEnergy;

    //The following lines are adapted from SiDigitization's Inserter class
    SiSurfaceCharge scharge(chargePos,SiCharge(ed,hitTime(phit),SiCharge::track,HepMcParticleLink(phit->trackNumber(),phit.eventId())));

     SiCellId diode = Module.cellIdOfPosition(scharge.position());

       SiCharge charge = scharge.charge();

      if (diode.isValid()) {

      chargedDiodes.add(diode,charge);
    }
  }
```

for #1

for #2

PixelBarrelChargeTool::charge(...) /2

- Type Conversion
- Variable declaration inside loop
- Alternate method available

```
for(int istep =0; istep < nsteps; istep++) {
  double xEta1 = xEta +  stepEta * (istep + 0.5);
  double xPhi1 = xPhi +  stepPhi * (istep + 0.5);
  double depD  = xDep +  stepDep * (istep + 0.5);

  // Distance between charge and readout side.  p_design->readoutSide() is
  // +1 if readout side is in +ve depth axis direction and visa-versa.
  double spess = 0.5 * sensorThickness - Module.design().readoutSide() * depD;
  if (spess<0) spess=0;

  for(int i=0 ; i<ncharges ; i++) {
    // diffusion sigma
    double rdif=this->m_diffusionConstant*sqrt(spess*coLorentz/0.3);
    // position at the surface
    double xPhiD=xPhi1+spess*tanLorentz+rdif*CLHEP::RandGaussZiggurat::shoot(m_rndmEngine);
    double xEtaD=xEta1+rdif*CLHEP::RandGaussZiggurat::shoot(m_rndmEngine);

    // Get the charge position in Reconstruction local coordinates.
    SiLocalPosition chargePos = Module.hitLocalToLocal(xEtaD, xPhiD);

    // The parametrization of the sensor efficiency (if needed)
    double ed=-1*this->electronHolePairsPerEnergy;

    //The following lines are adapted from SiDigitization's Inserter class
    SiSurfaceCharge scharge(chargePos,SiCharge(ed,hitTime(phit),SiCharge::track,HepMcParticleLink(phit->trackNumber(),phit.eventId())));

     SiCellId diode = Module.cellIdOfPosition(scharge.position());

       SiCharge charge = scharge.charge();

      if (diode.isValid()) {

      chargedDiodes.add(diode,charge);
    }
  }
```

for #1

for #2

Expensive!

PixelBarrelChargeTool::charge(...) /3

- Some technical details...
  - CLHEP::RandGaussZiggurat::shoot(...) can be replaced by ::shootArray(...) outside of the loop
    - Means we have a double* assigned to the heap (and then de-allocated...)
    - Better to put on the stack? (less scale safe, and we get a warning)
    - Can also have a frozen size array, with some safety mechanism to prevent overflow

```
for(int istep =0; istep < nsteps; istep++) {
  double xEta1 = xEta0 +  stepEta * (istep + 0.5);
  double xPhi1 = xPhi0 +  stepPhi * (istep + 0.5);
  double depD  = xDep0 +  stepDep * (istep + 0.5);

  // Distance between charge and readout side.  p_design->readoutSide() is
  // =1 if readout side is in +ve depth axis direction and visa-versa.
  double spess = 0.5 * sensorThickness - Module.design().readoutSide() * depD;
  if (spess<0) spess=0;

  for(int i=0 ; i<ncharges ; i++) {
    // diffusion sigma
    double rdif=this->m_diffusionConstant*sqrt(spess*coLorentz/0.3);
    // position at the surface
    double xPhiD=xPhi1+spess*tanLorentz+rdif*CLHEP::RandGaussZiggurat::shoot(m_rndmEngine);
    double xEtaD=xEta1+rdif*CLHEP::RandGaussZiggurat::shoot(m_rndmEngine);

    // Get the charge position in Reconstruction local coordinates.
    SiLocalPosition chargePos = Module.hitLocalToLocal(xEtaD, xPhiD);

    // The parametrization of the sensor efficiency (if needed)
    double ed=-1*this->electronHolePairsPerEnergy;

    //The following lines are adapted from SiDigitization's Inserter class
    SiSurfaceCharge scharge(chargePos,SiCharge(ed,hitTime(phit),SiCharge::track,HepMcParticleLink(phit->trackNumber(),phit.eventId())));

     SiCellId diode = Module.cellIdOfPosition(scharge.position());

       SiCharge charge = scharge.charge();

      if (diode.isValid()) {

      chargedDiodes.add(diode,charge);
    }
  }
```

for #1

for #2

Expensive!

Summary

- **Digitization code seems to be better suited to sub-event parallelism rather than having multiple events in flight.**
- **High I/O (the price we chose to avoid high memory) is definitely an issue now.**
- **Huge number of services created, not clear that these can be shared between threads.**
- **Alternative approaches being tested.**
- **HepMCParticleLink and xAOD::EventInfo migrations on-going.**
- **Code clean-up on-going**

IPileUpTool

```
0014 #include "EventInfo/PileUpEventInfo.h"
0015 #include "xAODEventInfo/EventInfo.h"
0016
0017 #include <vector>
0018 //typedef PileUpEventInfo::SubEvent::const_iterator SubEventIterator;
0019 typedef std::vector<xAOD::EventInfo::SubEvent>::const_iterator SubEventIterator;
0020
0021 class IPileUpTool : virtual public IAlgTool{
0022 public:
0023   ///called before the bunchXing loop
0024   virtual StatusCode prepareEvent(unsigned int /*nInputEvents*/) { return StatusCode::SUCCESS; }
0025   ///called for each active bunch-crossing (time in ns)
0026   virtual StatusCode processBunchXing(int bunchXing,
0027                                       SubEventIterator bSubEvents,
0028                                       SubEventIterator eSubEvents) = 0;
0029   ///flags whether this tool is "live" for bunchXing (time in ns)
0030   /// implemented by default in PileUpToolBase as FirstXing<=bunchXing<=LastXing
0031   virtual bool toProcess(int bunchXing) const =0;
0032   ///called at the end of the bunchXing loop
0033   virtual StatusCode mergeEvent() { return StatusCode::SUCCESS; }
0034   ///alternative interface which uses the PileUpMergeSvc to obtain all
0035   ///the required SubEvents.
0036   virtual StatusCode processAllSubEvents() = 0;
0037   ///flags whether the event should be removed or not
0038   virtual bool filterPassed() const =0;
0039   ///reset the filter
0040   virtual void resetFilter() =0;
0041
0042   static const InterfaceID& interfaceID() {
0043     static const InterfaceID IID( "IPileUpTool", 1, 0 );
0044     return IID;
0045   }
0046 };
```

IBkgStreamsCache

```
class IBkgStreamsCache : virtual public IAlgTool {
public:
  /**
      @param firstXing index of first xing to be processed (0=t0)
      @param nXings number of bunch Xings to be processed
      @param firstStore id of first store in cache
  */
  virtual StatusCode setup(int firstXing,
                 unsigned int nXings,
                 unsigned int firstStore,
                 IBeamIntensity* iBM)=0;
  /// inform concrete cache that we start overlaying a new event
  virtual void newEvent() = 0;
  /// rescale number of events per crossing
  virtual void resetEvtsPerXingScaleFactor(float sf) = 0;
  /**
     @brief Read input events in bkg stores and link them to overlay store
     @param iXing         offset to first xing number (=0 first Xing, =nXings for last xing)
     @param overlaidEvent reference to resulting overlaid event
     @param t0BinCenter   time wrto t0 of current bin center in ns
     @param BCID          bunch-crossing ID of signal bunch crossing
     @param loadEventProxies should we load the event proxies or not.
  */
  virtual StatusCode addSubEvts(unsigned int iXing,
                 PileUpEventInfo& overEvent,
                 int t0BinCenter, bool loadEventProxies, unsigned int BCID) = 0;
  /**
     @brief Read input events in bkg stores and link them to overlay store
     @param iXing         offset to first xing number (=0 first Xing, =nXings for last xing)
     @param overlaidEvent reference to resulting overlaid event
     @param t0BinCenter   time wrto t0 of current bin center in ns
  */
  virtual StatusCode addSubEvts(unsigned int iXing,
                 PileUpEventInfo& overlaidEvent,
                 int t0BinCenter) = 0;
  /// how many stores in cache
  virtual unsigned int nStores() const = 0;

  static const InterfaceID& interfaceID() {
    static const InterfaceID IID( "IBkgStreamsCache", 1, 0 );
    return IID;
  }
};
```