lcsim PFA guide

Guide for Particle Flow Algorithm developers in org.lcsim

This page documents the framework for developing PFA algorithms, explains the conventions used, and gives example implementations.

- #Conventions
- #Worked examples
    - #HitMap manipulation
    - #A very trivial PFA
    - #Using DigiSim
    - #Reading in and writing out hitmaps
    - #How to make things appear in WIRED or the Event Browser in JAS3
- #Outline of a complete PFA
- #Pieces of a PFA
- #Things that need doing

Conventions

In order to make the PFA components as interchangeable as possible, we have adopted some conventions. These were discussed at the January 2005 Boulder simulation workshop. They will probably evolve slowly over time.

- PFAs should be structured as a series of Drivers. This way, components can be swapped in and out easily.
- When PFA Drivers need to communicate with each other, they should do it by storing objects or collections of objects in the event via the put() and get() methods.
- Groups of hits should be stored as Hitmaps. The HitMap class is an extension of HashMap<Long,CalorimeterHit>. Each entry corresponds to one hit, giving the CellID and the hit itself.
- Often, a Driver takes a HitMap as input, performs some operation on it, and produces a modified version of the HitMap as part of the output. The output HitMap should be a new, separate object (i.e. the original HitMap should be left unchanged at the end). Example code to read in and write out hitmaps is given below.
- Clusters should be stored in the event as List<Cluster>.

Worked examples

Here are some example code snippets showing how to combine drivers to produce a PFA. **These are written to be read by people rather than compilers**, so they may need extra tweaks to run in practice. If you know of other patterns, or if you see that these examples have become out of date, please update them.

HitMap manipulation

There are several classes in the org.lcsim.util.hitmap package which can be used to manipulate HitMaps. These are generally wrapped as Drivers so that they can be dropped into a PFA template. Here is a quick list:

- Add, subtract, or clone a named HitMap in the event via a driver:
    - HitMapAddDriver
    - HitMapSubtractDriver
    - HitMapCloneDriver
- Convert formats (including the previous Map<Long,CalorimeterHit> format for backwards-compatability):
    - HitMapToClusterListDriver
    - HitMapToHitListDriver
    - MapToHitMapDriver
    - HitListToHitMapDriver
    - ClusterListToHitMapDriver
- Filter the hits in a hitmap:
    - HitMapFilter
    - HitMapFilterDriver

A very trivial PFA

See TrivialPFA.java in CVS for a worked implementation. TrivialPFA can be accessed from the examples page within JAS, though you will have to compile and load it manually. Here is an even simpler piece of code:

```
import org.lcsim.util.hitmap.*;
import org.lcsim.event.*;
import org.lcsim.event.util.*;
import org.lcsim.recon.cluster.cheat.PerfectClusterer;

public class SimplePFA extends Driver
{
  public SimplePFA()
  {
    // Set up a hitmap to hold the raw calorimeter hits in the event
    // This driver reads in a bunch of List<CalorimeterHit> and writes
    // out a HitMap to the event.
    HitListToHitMapDriver rawHitMap = new HitListToHitMapDriver();
    rawHitMap.addInputList("EcalBarrHits");
    rawHitMap.addInputList("EcalEndcapHits");
    rawHitMap.addInputList("HcalBarrHits");
    rawHitMap.addInputList("HcalEndcapHits");
    rawHitMap.setOutput("raw hitmap");
    add(rawHitMap);

    // Set up a list of final-state Monte Carlo particles
    // This driver will write out a List<MCParticle> to the event.
    CreateFinalStateMCParticleList mcListMaker = new CreateFinalStateMCParticleList("Gen");
    add(mcListMaker);

    // Cluster the hits (perfect pattern recognition)
    // This driver will reads in the hitmap and the list of MCParticles.
    // It writes out a modified hitmap and a List<Cluster>.
    PerfectClusterer clusterer = new PerfectClusterer();
    clusterer.setInputHitMap("raw hitmap");
    clusterer.setOutputHitMap("leftover hits");
    clusterer.setOutputClusterList("perfect clusters");
    clusterer.setMCParticleList("GenFinalStateParticles");
    add(clusterer);

    // [The rest of the PFA would go here]
  }
}
```

Using DigiSim

Thanks to Guilherme, we have a digitisation simulation package called DigiSim which is available under org.lcsim.digisim. There is an example driver at org.lcsim.plugin.web.examples.DigiSimExample. Borrowing heavily from that, here is an example snippet showing how to use the output from DigiSim:

```
public class SimplePFA extends Driver
{
  public SimplePFA()
  {
    // CalHitMapDriver is a driver that produces hitmaps in the format
    // needed by DigiSim:
    add(new org.lcsim.recon.cluster.util.CalHitMapDriver());
    // Run DigiSim, producing raw hit collections:
    org.lcsim.digisim.DigiSimDriver digi = new org.lcsim.digisim.DigiSimDriver();
    add(digi);
    // Convert the output to SimCalorimeterHit format for use in analysis:
    add( new org.lcsim.digisim.SimCalorimeterHitsDriver() );

    // Now we can add some more drivers to analyze the output. For example:

    // Set up a hitmap for the digisim output hits
    HitListToHitMapDriver digiHitMap = new HitListToHitMapDriver();
    digiHitMap.addInputList("EcalBarrDigiHits");
    digiHitMap.addInputList("EcalEndcapDigiHits");
    digiHitMap.addInputList("HcalBarrDigiHits");
    digiHitMap.addInputList("HcalEndcapDigiHits");
    digiHitMap.setOutput("digi hitmap");
    add(digiHitMap);

    // Set up the MC list
    CreateFinalStateMCParticleList mcListMaker = new CreateFinalStateMCParticleList("Gen");
    add(mcListMaker);

    // Cluster the hits (perfect pattern recognition)
    PerfectClusterer clusterer = new PerfectClusterer();
    clusterer.setInputHitMap("digi hitmap");
    clusterer.setOutputHitMap("leftover hits");
    clusterer.setOutputClusterList("perfect clusters");
    clusterer.setMCParticleList("GenFinalStateParticles");
    add(clusterer);
  }
}
```

Reading in and writing out HitMaps

If you are writing your own drivers to use in a PFA, you will probably need to read in HitMaps, and maybe also write out a modified HitMap. Here is a snippet showing how to do this.

```
public class HitMapReader extends Driver
{
  public HitMapReader(String inputName, String outputName)
  {
    m_inputName = inputName;
    m_outputName = outputName;
  }

  public void process(EventHeader event)
  {
    // Read in the hitmap with the given name from the event.
    HitMap inputHitMap = (HitMap) (event.get(m_name));
    // Now produce a clone so we can write it out later:
    HitMap outputHitMap = new HitMap(inputHitMap); // initially cloned

    // [Do some manipulation here, e.g. making clusters and removing their hits from outputHitMap]

    // Example: Here's one way to loop over the hits in the hitmap:
    for (CalorimeterHit hit : inputHitMap.values()) {
      System.out.println("Here is a hit: "+hit);
    }

    // Write out the hitmap:
    event.put(m_outputHitMapName, outputHitMap);
  }

  String m_inputName;
  String m_outputName;
}
```

How to make things appear in WIRED or the Event Browser in JAS3

One nice feature of JAS3 is that when you open a file in org.lcsim mode, you can upload stuff to the event and have it appear in the org.lcsim Event Browser and the WIRED display (see: Displaying analysis objects with wired). But there are a few tricks to make this work:

- For the Event Browser, there must be a handler class in org.lcsim.plugin.browser to tell it how to display the table. Otherwise, the thing you uploaded will appear in the list but won't display anything useful.
- For WIRED, you also need a handler class or the thing you uploaded won't be visible.
- Currently, these expect things to be uploaded as a List<Object>. So if the objects are in another format – such as a hitmap – they won't be readable.

Here is a code snippet that allows you to display hitmaps:

```
import org.lcsim.util.hitmap.HitMapToHitListDriver;

// Here is the PFA class:
public class SimplePFA extends Driver
{
  public SimplePFA()
  {
    // [drivers go here, producing a HitMap called "digi hitmap"]

    // Here's the driver to convert the hitmap into a List<CalorimeterHit> to display:
    HitMapToHitListDriver digiConverterDriver = new HitMapToHitListDriver();
    digiConverterDriver.setInputHitMap("digi hitmap");
    digiConverterDriver.setOutputList("digi hits (displayable)");
    add(digiConverterDriver);

    // [rest of the code goes here]
  }
}
```

Outline of a complete PFA

Based on the discussions at Boulder, here is an outline (very abstracted!) of what a real PFA might look like.

```
public class CompletePFA extends Driver
{
  public CompletePFA()
  {
    // First, use DigiSim to make more realistic hits
    add(new org.lcsim.recon.cluster.util.CalHitMapDriver());
    org.lcsim.digisim.DigiSimDriver digi = new org.lcsim.digisim.DigiSimDriver();
    add(digi);
    add( new org.lcsim.digisim.SimCalorimeterHitsDriver() );

    // Produce hitmaps:
    HitListToHitMapDriver digiHitMap = new HitListToHitMapDriver();
    digiHitMap.addInputList("EcalBarrDigiHits");
    digiHitMap.addInputList("EcalEndcapDigiHits");
    digiHitMap.addInputList("HcalBarrDigiHits");
    digiHitMap.addInputList("HcalEndcapDigiHits");
    digiHitMap.setOutput("digi hitmap");
    add(digiHitMap);

    // Find tracks with the fast MC (output is a List<Track> saved as EventHeader.TRACKS)
    add (new org.lcsim.mc.fast.tracking.MCFastTracking());

    // Run a MIP-finder, possibly taking the tracks as input.
    // "MipFinder" is a made-up class.
    MipFinder exampleMipFinder = new MipFinder();
    exampleMipFinder.setInputTrackList(EventHeader.TRACKS);
    exampleMipFinder.setInputHitMap("digi hitmap");
    exampleMipFinder.setOutputClusterList("mips");
    exampleMipFinder.setOutputHitMap("digi hitmap after removing mips");
    add(exampleMipFinder);

    // Find E/M clusters
    // "EMFinder" is a made-up class.
    EMFinder exampleEMFinder = new EMFinder();
    exampleEMFinder.setInputHitMap("digi hitmap after removing mips");
    exampleEMFinder.setOutputClusterList("em showers");
    exampleEMFinder.setOutputHitMap("digi hitmap after removing mips and em showers");
    add(exampleEMFinder);

    // Identify the E/M clusters -- photons? electrons? pi0?
    // Output is a List<ReconstructedParticle>.
    // In reality we'd probably iterate a little here on the hit assignments,
    // and might need to pick up MIP segments for a few electrons, but neglect that for now.
    // "EMIdentifier" is a made-up class.
    EMIdentifier exampleEMIdentifier = new EMIdentifier();
    exampleEMIdentifier.setInputClusterList("em showers");
    exampleEMIdentifier.setInputTrackList(EventHeader.TRACKS);
    exampleEMIdentifier.setOutputParticleList("identified em particles");

    // Now find remaining clusters, which should mostly be from hadrons (and muons)
    // after a shower/interaction/scatter. This step is very abstracted and would
    // include all kinds of things such as fragment-handling.
    // "HADClusterer" is a made-up class.
    HADClusterer exampleHADClusterer = new HADClusterer();
    exampleHADClusterer.setInputHitMap("digi hitmap after removing mips and em showers");
    exampleHADClusterer.setInputMipList("mips");
    exampleHADClusterer.setInputTrackList(EventHeader.TRACKS);
    exampleHADClusterer.setOutputHitMap("digi hitmap after removing mips, em showers, and had clusters");
    exampleHADClusterer.setOutputClusterList("had");

    // Identify the hadronic/muon particles found:
    // "HADIdentifier" is a made-up class.
    HADIdentifier exampleEMIdentifier = new EMIdentifier();
    exampleHADIdentifier.setInputClusterList("had");
    exampleHADIdentifier.setInputTrackList(EventHeader.TRACKS);
    exampleHADIdentifier.setOutputParticleList("identified had particles");

    // Then we do something useful with all these ReconstructedParticles.
    // [analysis]
  }
}
```

Caveat: That is not real, compilable code! The real thing will not look exactly like it, since several problems were quietly swept under the carpet. But it illustrates the general structure.

Pieces of a PFA

One of the main advantages of a modular approach is that algorithms can be shared and re-used with minimal coding. In this section we'll look at some of the pieces of a PFA and show how they can be implemented with existing classes.

- Photon-finding
- MIPs and track segments
- Generic clusterers

Here are some classes that are in CVS and that slot into this framework. *(Note: This list is definitely not complete – it's just some of the ones I'm familiar with.)*

When writing classes, please strive for flexibility and reuseability.

- org.lcsim.recon.cluster.nn.NearestNeighborClusterDriver: This is a general-purpose clusterer. It uses a nearest-neighbor approach, starting from seed points and recursively adding nearby hits, where "nearby" is defined in terms of numbers of cells in each direction. The driver looks up every List<CalorimeterHit> in the event and applies the clusterer to each, writing out the clustered hits as a List<Cluster>. The name of each output list is the name of the input list plus a common extension (default is "NearestNeighborClusterDriver").
- org.lcsim.recon.cluster.mst.MSTClusterDriver: This is another general-purpose clusterer. It also works with a nearest-neighbor approach, but with a user-definable metric. The default metric is 3D distance between the centers of the two hit cells. This is in general more computationally expensive than the NN clusterer, but is less sensitive to changes in the detector geometry. The driver also has hooks to let the user veto certain kinds of hits or clusters.
- org.lcsim.recon.cluster.mipfinder.MIPClusterDriver: This clusterer is intended to find MIPs and track segments. It looks for sequences of isolated or near-isolated hits in adjacent layers.
- org.lcsim.recon.cluster.clumpfinder.ClumpFinder: This driver reads in a HitMap and tries to find dense clumps of hits. These are written out as a list of Clusters, and left-over hits are written out as a HitMap.

Here is some example code showing how these pieces can be combined. Notice how quite a bit of the code is just manipulation of HitMaps and cluster lists.

```
public class ExamplePFA extends Driver
{
  public ExamplePFA()
  {
    // [Set up the hit maps and run DigiSim as usual]

    // Find MIP and track segments in the ECAL and HCAL
    TrackClusterDriver ecalMIP = new TrackClusterDriver("input hit map ecal",
        "mips ecal", "hit map ecal without mips");
    TrackClusterDriver hcalMIP = new TrackClusterDriver("input hit map hcal",
        "mips hcal", "hit map hcal without mips");
    add(ecalMIP);
    add(hcalMIP);
    // Merge the two lists:
    ListAddDriver<Cluster> mergeMIPs = new ListAddDriver<Cluster>();
    mergeMIPs.addInputList("mips ecal");
    mergeMIPs.addInputList("mips hcal");
    mergeMIPs.setOutputList("mips");
    add(mergeMIPs);

    // Find photons in the ECAL (cheating here)
    add(new ListFilterDriver(new ParticlePDGDecision(22), mcListName, "MCParticles photons only"));
    PerfectClusterer myCheatPhotonFinder = new PerfectClusterer();
    myCheatPhotonFinder.setInputHitMap("hit map ecal without mips");
    myCheatPhotonFinder.setOutputHitMap("hit map ecal without mips or photons");
    myCheatPhotonFinder.setOutputClusterList("photon clusters");
    myCheatPhotonFinder.setMCParticleList("MCParticles photons only");
    myCheatPhotonFinder.allowHitSharing(false);
    add(myCheatPhotonFinder);

    // Find clumps in ECAL and HCAL
    ClumpFinder findClumpsECAL = new ClumpFinder("hit map ecal without mips or photons",
        "clumps ecal", "hit map ecal without mips or photons or clumps");
    ClumpFinder findClumpsHCAL = new ClumpFinder("hit map hcal without mips",
        "clumps hcal", "hit map hcal without mips or clumps");
    add(findClumpsECAL);
    add(findClumpsHCAL);
    ListAddDriver<Cluster> mergeClumps = new ListAddDriver<Cluster>();
```

```
    mergeClumps.addInputList("clumps ecal");
    mergeClumps.addInputList("clumps hcal");
    mergeClumps.setOutputList("clumps");
    add(mergeClumps);

    // Merge clumps, MIPs and any other hits into larger clusters.
    // We still retain the old lists, so we can go back later and
    // study the structure.

    // First, the ECAL:
    MSTClusterDriver mstEcal = new MSTClusterDriver("ecal hit map after mst", "mst clusters ecal");
    mstEcal.addInputHitMap("hit map ecal without mips or photons or clumps");
    mstEcal.addUserInputList("mips ecal");
    mstEcal.addUserInputList("clumps ecal");
    mstEcal.setThreshold(30.0); // 30mm point-to-point
    mstEcal.registerMetrics(new MinimumHitToHitDistance());
    add (mstEcal);

    // Then the HCAL:
    MSTClusterDriver mstHcal = new MSTClusterDriver("hcal hit map after mst", "mst clusters hcal");
    mstHcal.addInputHitMap("hit map hcal without mips or clumps");
    mstHcal.addUserInputList("mips hcal");
    mstHcal.addUserInputList("clumps hcal");
    mstHcal.setThreshold(100.0); // 100mm point-to-point
    mstHcal.registerMetrics(new MinimumHitToHitDistance());
    add (mstHcal);

    // Link clusters across the ECAL-HCAL boundary
    MSTClusterDriver mstDriverLink = new MSTClusterDriver("User");
    mstDriverLink.registerMetrics(new MinimumHitToHitDistance());
    mstDriverLink.setThreshold(50.0); // 50mm
    mstDriverLink.addUserInputList("mst clusters ecal");
    mstDriverLink.addUserInputList("mst clusters hcal");
    mstDriverLink.setClusterName("mst clusters linked");
    mstDriverLink.setPairDecision(new BothCalorimetersDecision());
    add(mstDriverLink);
  }
}
```

Things that need doing

Updates needed in the code:

- A real, live PFA in this format
- Standard routines for telling you how well your PFA did (Ron's ClusterAnalysis?)
- Example(s) in the main org.lcsim tree, probably under org.lcsim.plugin.web.examples

Updates needed on this page:

- Detail on the various PFA components (MIP finder, photon finder, hadron clusterer, cluster identifier, ...) – probably that belongs in sub-pages.
- Link to relevant things
- More code examples