IST offline software

Yaping Wang, Zhenyu Ye
(University of Illinois at Chicago)

1. Introduction

The Heavy Flavor Tracker (HFT) of the STAR experiment is designed to improve track impact parameter resolution to enable measurement of secondary vertices from heavy flavor hadron decays. The HFT consists of three sub-detectors: a silicon pixel detector, an Intermediate Silicon Tracker (IST), and a silicon strip detector. This note describes the offline reconstruction chain of the IST in the STAR big full chain. In the following, I describe the IST offline software structure, data structures, reconstruction makers, and DB tables in section 2, section 3, section 4 and section 5, respectively.

2. IST Offline Software Structure

The IST offline software has been fully tested with commissioning cosmic data and Au+Au collision data at 15 GeV and 200 GeV. The IST offline software overall structure is shown as Fig. 1.

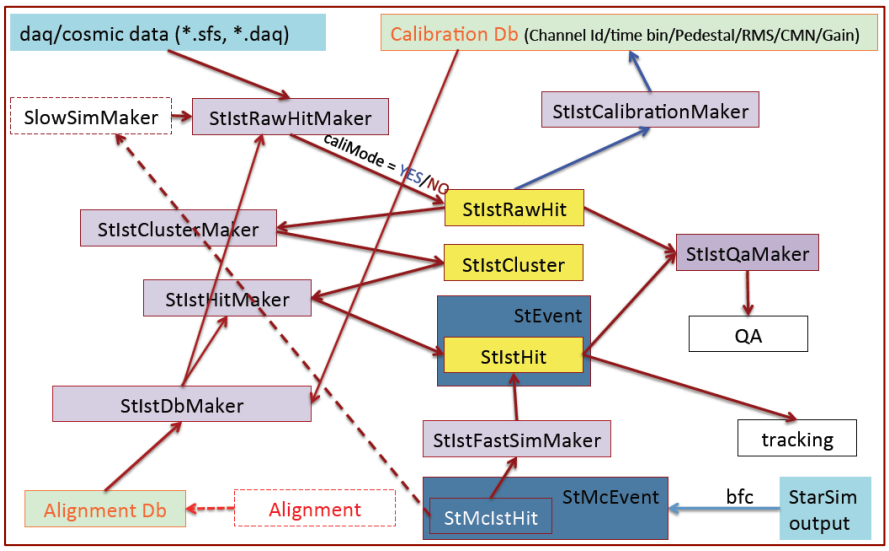

Fig. 1 The IST software infrastructure

The offline chain processes raw data (both non-Zero suppression and Zero Suppression data) and reconstructs IST hit within STAR BFC chain for STAR tracking. As shown in Fig. 1, it includes the following major parts: Raw hit maker, cluster maker, hit maker, DB maker, calibration maker, QA maker and simulation makers (Fast simulator is available, and the slow simulator is under developing). Three kinds of data structure are designed for three levels of IST raw hit, the IST cluster and the IST hit, respectively. In addition, offline calibration/geometry DB tables are constructed for the hit reconstruction.

StIstRawHitMaker un-packs the IST signal (such as RDO Id, ARM Id, APV Id, channel Id, time bin index, ADC) for all channels every event from the raw data,

and it has two working modes: One is calibration mode, and the other mode is physics mode. In calibration mode, it processes non-zero suppression data only and decodes the raw data into raw hit structure (StIstRawHit). In the later mode, calibration DBs are accessed by StIstDbMaker to obtain mean pedestal, RMS noise, common-mode noise, mapping, control parameters for signal calibrations.

StIstClusterMaker does clustering for neighboring raw hits, and loops the raw hit collections (StIstRawHitCollection) as input. The generated clusters will be stored in cluster collections (StIstClusterCollection). Currently, two clustering algorithms (StIstSimpleClusterAlgo and StIstScanClusterAlgo) are available, and they will be described in detailed in following section.

StIstHitMaker retrieves each hit's information from the cluster collections, and calculates its global position. The geometry Dbs are accessed via the StIstDbMaker for the hit global position calculation. The maker outputs the IST hits and stores them to StEvent for tracking.

StIstDbMaker gets handlers of geometry and calibration Dbs from StarDb.

StIstCalibrationMaker calculates pedestal, rms noise and common mode noise over all time bins. The maker produces two data files, which are used for writing calibration Dbs. The gain calculation for each channel will be available later, and currently we set the gain to 1.0 for all channels.

StIstFastSimMaker makes StMcIstHit directly from StarSim output for fast simulation. The StIstSlowSimMaker is not available yet, which will be designed to make raw hits for slow simulation.

StIstQaMaker is designed to generate QA histograms and trees for quick check on IST raw hit and hit levels.

Some basic constants for the IST detector are saved in the StIstConsts.h and StEnumerations.h. The whole IST offline software distributes in the StRoot directory, and will be listed in appendix 1.

3. IST raw hit/cluster/hit structures

There are three kinds of data structure for the IST raw hit, cluster and hit, respectively. For these data structures, corresponding containers are developed.

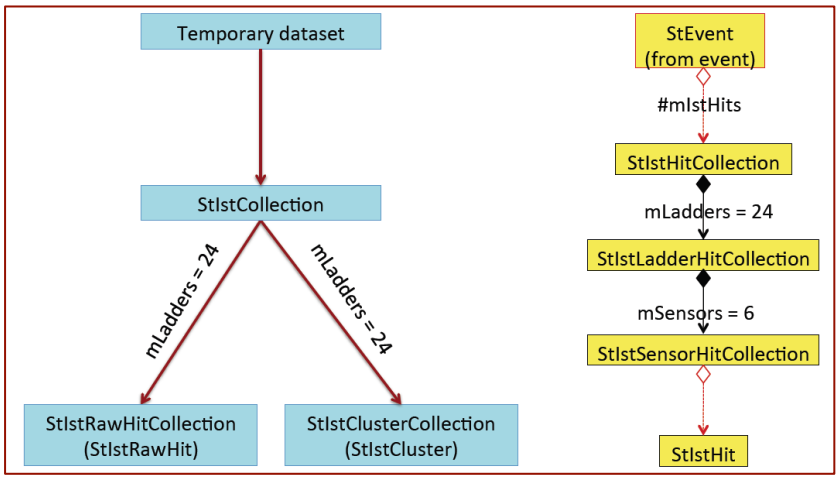

Fig. 2 IST raw hit, cluster and hit containers and hierarchy structure

The IST raw hit information and cluster information are only survived in the period of reconstruction chain, while the IST hit information will be written to StEvent serving the whole STAR tracking combined with hits information from other sub-detectors. The overall storage scheme of the IST raw hit, cluster and hit containers is shown in Fig. 2.

3.1 StIstRawHit/StIstRawHitCollection

Each IST raw hit corresponds to a fired silicon pad (channel), and the following data members are defined for the IST raw hit:

```
Int_t      mChannelId;                    // channel Id, numbering from 0 to 110591
Int_t      mGeoId;                        // geometry Id, numbering from 1 to 110592
Float_t    mCharge[kIstNumTimeBins];// raw ADC value saved in calibration mode;
    // charge value saved in non-calibration mode (pedestal subtracted and CMN corrected)
Float_t    mChargeErr[kIstNumTimeBins]; // charge error in all time bins
UChar_t        mMaxTimeBin;       // the max ADC time bin index of the raw hit
UShort_t       mIdTruth;          // !< for embedding, 0 as background
static UChar_t mDefaultTimeBin;   // default max ADC time bin index
```

The IST raw hits are stored in the StIstRawHitCollection container sorted by each raw hit geometry ID in ascending order. The raw hit geometry ID is retrieved from IST offline mapping DB table, which is produced by readout electronics mapping. The ladder ID, sensor ID, column index and row index of the raw hit will be calculated via the obtained geometry Id (mGeoId) in StIstRawHit. The StIstRawHitCollection will be saved to a temporary dataset, called StIstCollection as shown in Fig. 2.

3.2 StIstCluster/StIstClusterCollection

Each IST cluster is a group of neighboring raw hits, and following data members are defined for the IST cluster:

```
Int_t      mKey;                          //cluster unique label
UChar_t    mLadderId;                     //ladder id the cluster belongs to
UChar_t    mSensorId;                     //sensor id the cluster belongs to
Float_t    mMeanRow, mMeanColumn;         //mean row and mean column
Float_t    mTotCharge;                    //charge sum of the cluster
Float_t    mTotChargeErr;                 //RMS noise of the cluster
UChar_t    mMaxTimeBin;                   //max ADC time bin index
UChar_t    mClusteringType;               //clustering algorithm type
UChar_t    mNRawHits;                     //cluster size
UChar_t    mNRawHitsRPhi;                 //cluster size in r-phi direction
UChar_t    mNRawHitsZ;                    //cluster size in beam direction
UShort_t   mIdTruth;                      //!< for embedding, 0 as background
std::vector<StIstRawHit *> mRawHitVec;    //vector container to save raw hits
                                          //who contribute to the cluster
```

The IST clusters are stored in the StIstClusterCollection container, which is saved to the StIstCollection as shown in Fig. 2. There is a common container designed for the StIstRawHitCollection and StIstClusterCollection, and these raw hit and cluster collections are arranged in ladder level.

The IST raw hit/cluster data structures and their containers locate under the StIstUtil directory.

3.3 StIstHit/StIstHitCollection

Each IST hit is generated from an IST cluster, and the hit global position is included. The StIstHit is inherited from StHit basic class, and the following specified data members are defined:

```
UChar_t    mMaxTimeBin;          // max charge time bin
Float_t    mChargeErr;           // charge uncertainty
UChar_t    mNRawHits;            // nRawHits: cluster size
UChar_t    mNRawHitsZ;           // cluster size in Z direction
UChar_t    mNRawHitsRPhi;        // cluster size in r-phi direction
Float_t    mLocalPosition[3];    // local position of hit inside the sensor
StDetectorId mDetectorId;        // kIstId
```

As shown in Fig. 2, the IST hit is stored in the three-level hierarchy containers in top-bottom way: StIstHitCollection → StIstLadderHitCollection → StIstSensorHitCollection.

Finally, the StIstHitCollection container is saved to StEvent. To avoid external constants accessing in StIstHit, a specified class StIstDigiHit was developed. The class inherits from StIstHit, and it provides mean of getting APV ID, mean column, mean row and uncertainties of local positions.

4. IST raw hit/cluster/hit makers

4.1 StIstRawHitMaker

StIstRawHitMaker un-packs the daq and/or sfs format data and produces IST raw hits. The IST daq(sfs) data contains electronics information, such as **rdo** (ARM Readout Controller board ID, numbering from 1 to 6 per whole IST readout system), **arm** (APV readout module board ID, numbering from 0 to 5 per rdo unit), **apv** (APV chip ID, numbering from 0 to 23 per arm unit), **channel** (readout channels ID, numbering from 0 to 127 per APV chip unit), **adc** (ADC value, range from 0 to 4095) and **time bin** (time bin index, numbering from 0 to 4/5/9/15/31). Meanwhile, the broken/noisy channels will be masked out by extreme high RMS noise settings (100.00 ADC counts, this value is much higher than nominal value ~20.0 ADC counts). The bad/strange chips will be masked out by high common mode noise settings (100.00 ADC counts, this value is much higher than nominal value ~ 10.0 ADC counts). The StIstRawHitMaker has two working modes: One is Calibration mode, and the other is Physics mode.

In calibration mode, the maker deals with non-ZS data only. The raw ADCs in

of time bins with the corresponding channel electronics ID are stored to the StIstRawHitCollection as input of IST calibration maker.

In physics mode, both non-ZS and ZS data are processed and packed in IST raw hit structure. In the physics mode, pedestal subtraction for non-ZS data are enabled. Signal-like raw hit decision is executed to exclude noise, and it requires three consecutive time bins' pedestal-subtracted ADC larger than 5 times of RMS noise. In addition, the dynamical common mode noise is calculated and the correction can be enabled/disabled. In this mode, the offline calibration DB tables are accessed via StIstDbMaker, including istPedNoise table, istMapping table, istControl table and istGain table. The Fig. 3 shows the ADC spectrum of all raw hits in time bin 0 (left) and time bin 1 (middle), and the RMS noise (right) of all raw hits. The left two narrow gaps are due to bias voltage problems, and the right wider gap is caused by the dead section (two sensors). Fig. 4 shows the IST raw hit map (left) and max time bin (right) as a function of readout electronics coordinates.

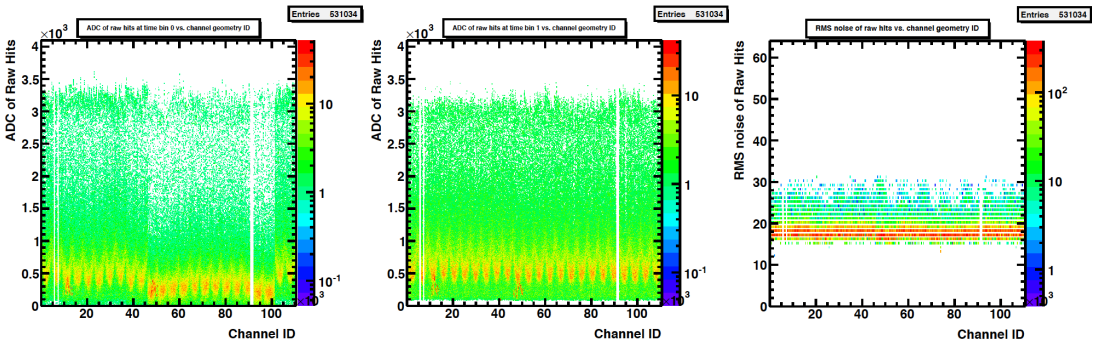

Fig. 3 ADC spectrum of IST raw hits at time bin 0 (left) and time bin 1 (middle); RMS noise of IST raw hits (right)

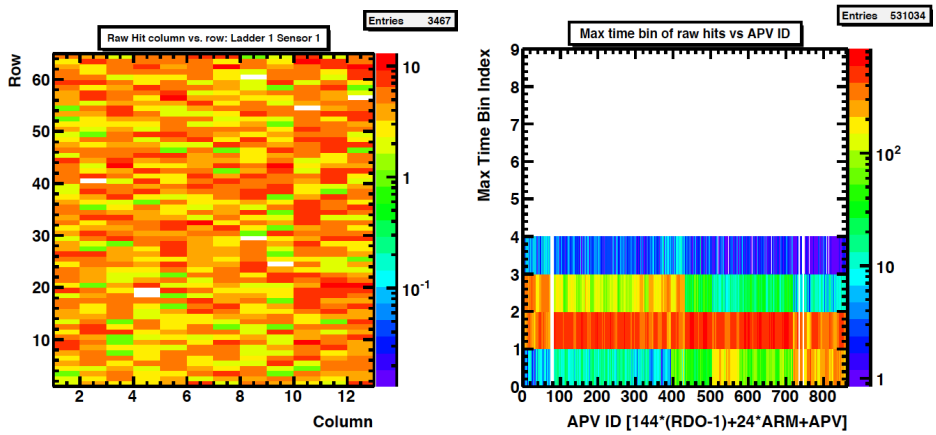

Fig. 4 IST raw hit map per sensor (left) and max ADC time bin (right)

The ADC to charge/energy translation was carried out in this mode via applying for the channel's gain (The gain is set to 1.0 for all channels).

4.2 StIstClusterMaker

The StIstClusterMaker loops the raw hit collections ladder by ladder, and executes the raw hits clustering over a sensor area sensor by sensor. The maker produces IST clusters and stores them into StIstClusterCollection.

An interface, "Int_t setClusterAlgo(StIstIClusterAlgo*)" is designed in the StIstClusterMaker to choose different clustering algorithms easily. Here, all algorithm classes are inherited from a virtual class StIstIClusterAlgo. The detailed clustering is done in selected clustering algorithm.

In the clustering algorithm, the algorithm calculates weighted ADC sum/Charge sum, weighted noise, and weighted column/row for each cluster. These parameters are calculated by following formulas:

$$ADC_{cluster} = \sum_{i=1}^{N_{pads}} ADC_i \quad , \quad \sigma_{cluster} = \sqrt{\sum_{i=1}^{N_{pads}} \sigma_i^2 / N_{pads}}$$

$$x_{cluster} = \sum_{i=1}^{N_{pads}} x_i \cdot w_i \quad , \quad y_{cluster} = \sum_{i=1}^{N_{pads}} y_i \cdot w_i$$

$$w_i = ADC_i / \sum_{i=1}^{N_{pads}} ADC_i$$

Here the N_{pads} represents cluster size. ADC_i and σ_i represent the pedestal subtracted ADC value and RMS noise for the ith raw hit of the current cluster, respectively. x_i and y_i represent mean row and mean column, respectively. The clustering algorithm is shown as below Fig. 5.

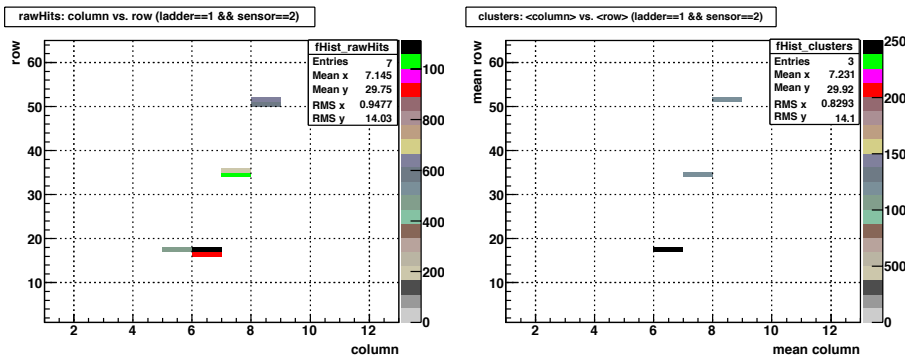

Fig. 5 IST clustering scheme for single event display (Left: Before clustering; Right: After clustering)

Two algorithms have been available for the raw hits clustering.

1) StIstSimpleClusterAlgo

- Read all raw hits per ladder (six sensors) and write into a vector sorted in an increasing order by geometry ID.
- Start from the 1st raw hit, and loop the vector to look for neighboring raw hits (in a sensor area) and do clustering. The found cluster will be filled into a ladder cluster collection.
- A case-by-case splitting algorithm can be enabled/disabled for the found clusters (here only works for cases with cluster size ≤ 4).

2) StIstScanClusterAlgo

- Read all raw hits per ladder (six sensors) and group into vectors (each vector is corresponding to a sensor column).
- Clustering in individual column: loop column vectors and scan each

vector sequentially to look for neighboring raw hits and do clustering. Once three continuous raw hits are found and the middle one has the minimum ADC/charge value, then the middle raw hit will be split into two parts weighted by its two neighboring raw hits' ADC/charge, respectively. The found cluster candidates will be filled into the corresponding cluster vector (each cluster vector is corresponding to a sensor column).

- Clustering between columns: loop neighboring columns, and do clustering once two cluster candidates are found with their weighted row index difference ($|\langle row\rangle_i - \langle row\rangle_j|$) less than 0.5.
- Fill the found clusters into the ladder cluster collection.

The performance comparison with the two algorithms can be found: http://www4.rcf.bnl.gov/~ypwang/IST_software/weeklyMeeting_Yaping.pdf

4.3 StIstHitMaker

StIstHitMaker generates IST hit based on the cluster information, and calculates hit's global position. The global position is calculated based on hit's local x/y/z by the rotations and translations according to the geometry DB tables retrieved via StIstDbMaker. The maker can process hit from the existed hit collection to re-calculate its global position or processes the cluster from the cluster collections to generate IST hits. The Fig. 6, 7 and 8 show the IST hit characteristics.

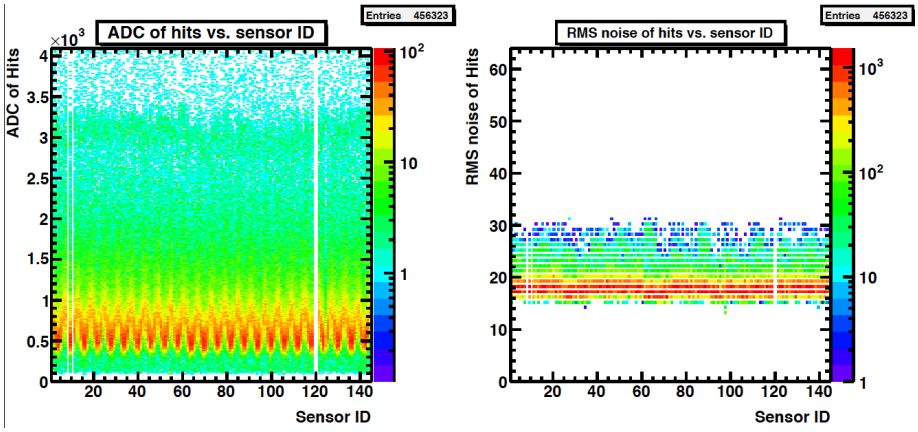

Fig. 6 ADC spectrum (left) and RMS noise (right) of IST hit

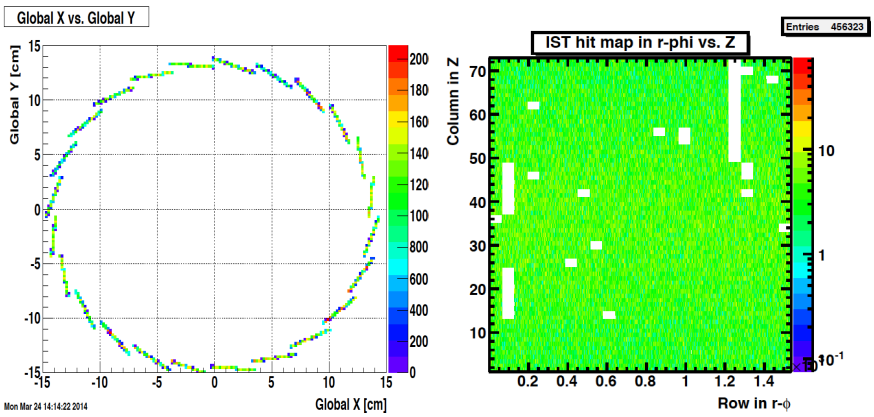

Fig. 7 IST hit Global position in east view (left) and hit map (right)

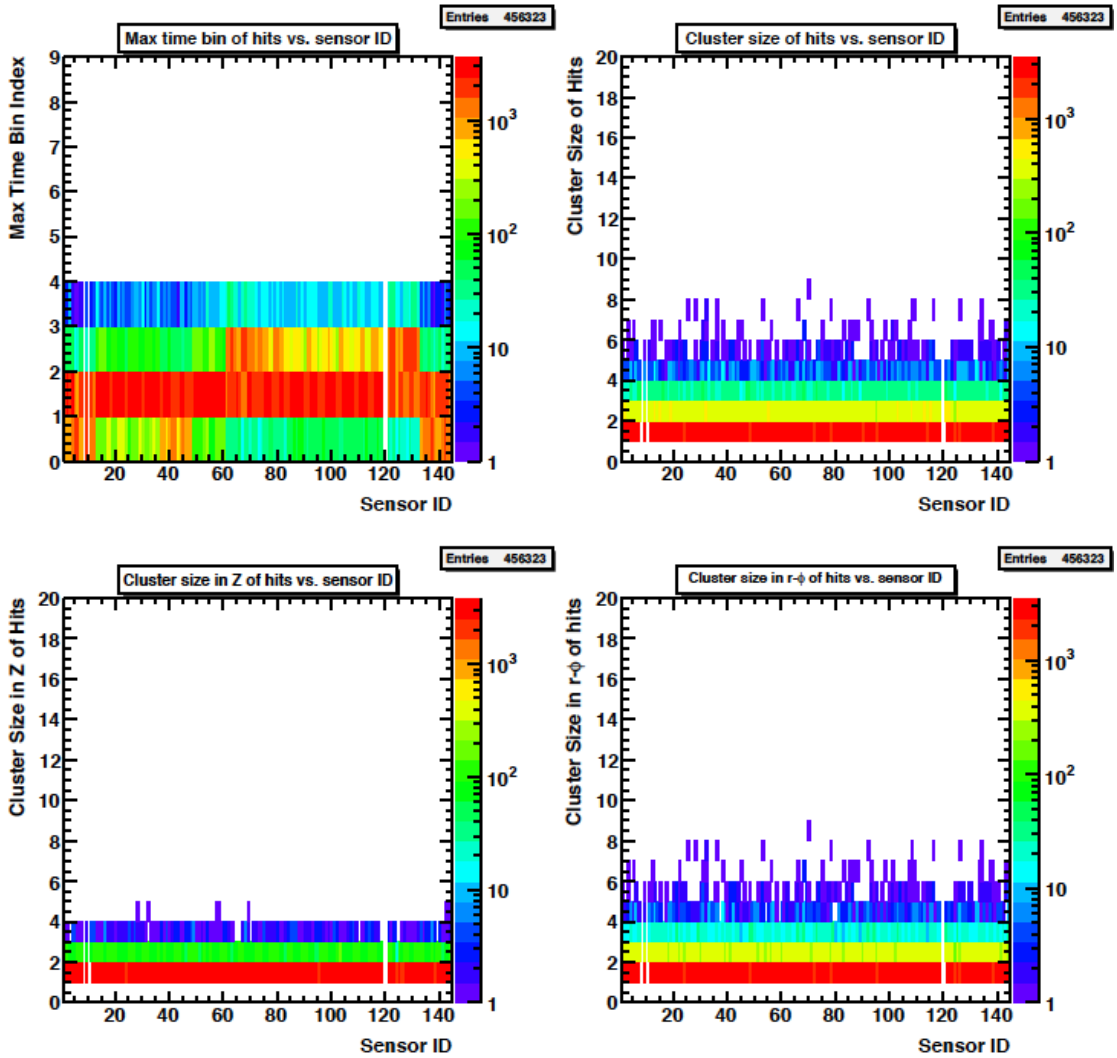

Fig. 8 IST hit max time bin (top-left), cluster size (top-right), cluster size in r-phi direction (bottom-left) and cluster size in Z direction (bottom-right)

The rotations and translations of each IST silicon sensor relative to the STAR global coordinate system are calculated in the StIstDbMaker, and this geometry information will be retrieved for the hit global position calculations.

5. IST geometry/calibration Db maker (StIstDbMaker)

The StIstDbMaker is designed to obtain geometry and calibration Db tables. Based on IST assembly structure, 5 geometry DB tables are constructed as described below:

idsOnTpc: translations/rotations of IDS relative to the TPC coordinate system, formatted as geometry matrix ids2Tpc.

pstOnIds: translations/rotations of PST relative to the IDS coordinate system, formatted as geometry matrix pst2Ids.

istOnPst: translations/rotations of whole IST detector relative to the PST coordinate system, formatted as geometry matrix ist2Pst.

istLadderOnIst: translations/rotations of IST ladder relative to the whole

IST detector's coordinate system, formatted as geometry matrix ladder2Ist.

istSensorOnLadder: translations/rotations of IST silicon sensor relative to the IST ladder coordinate system, formatted as geometry matrix sensor2Ladder.

The translations and rotations of IST silicon sensor relative to the STAR coordinate system can be calculated as below formula:

sensorGlobal = tpc2Global * ids2Tpc * pst2Ids * ist2Pst * ladder2Ist * sensor2Ladder

Here the tpc2Global represents the translations and rotations of TPC relative to the STAR global coordinate system. Two versions of geometry Db tables are available, one is populated with GEANT parameters and the other is populated with survey/alignment parameters. They are formatted in Survey_St structure and saved on STAR DB server.

There are four calibration tables defined for the IST offline reconstruction, including istMapping, istControl, istGain and istPedNoise.

istMapping: translation of channel electronics ID to geometry ID.

istControl: several pre-set cuts, such as hit cut, pedestal cut, noise thresholds, data types, dynamical number of time bin, and so on.

istGain: translation of IST signal from ADC counts to charge/energy.

istPedNoise: common-mode noise of all APV chips, pedestal and RMS noise of all channels. Dead/noisy channels or chips are masking by the extremely high RMS noise or CM noise setting (100.00 ADC counts).

Both the geometry and calibration Db tables are tagged with time stamp. The StIstDbMaker provides handler to access the above calibration datasets.

These tables can be checked at: http://online.star.bnl.gov/dbExplorer/

6. IST pedestal/noise maker (StIstCalibrationMaker)

The StIstCalibrationMaker calculates pedestal and RMS noise are calculated by histogram method and mathematical method for all channels over all time bins. The common-mode noise is calculated by histogram method for all chips over all time bins.

Mathematical method implemented in the calibration maker is executed in following steps:

(1) Accumulate event by event for each channel each time bin

```
p->ped[arm][apv][ch][tb] += (float) adc ;
p->rms[arm][apv][ch][tb] += (float) (adc * adc) ;
p->cou[arm][apv][ch][tb]++ ;
```

(2) Calculate pedestal/rms for each channel each time bin

```
pp   = ped->ped[arm][apv][ch][t]/(double) ped->cou[arm][apv][ch][t];
rr   = ped->rms[arm][apv][ch][t]/(double) ped->cou[arm][apv][ch][t];
rr   = sqrt(rr - pp*pp) ;
ped->ped[arm][apv][ch][t] = pp ;
ped->rms[arm][apv][ch][t] = rr ;
```

Histogram method implemented in the calibration maker is designed as below steps:

(1) Fill histograms event by event for each channel each time bin

```
int code = kIstNumTimeBins * elecId + t;
TH1F* histPed = mHistPedVec[ code ];
histPed->Fill( (float)adc );
```

(2) Exclude possible signal entries for each channel each time bin

```
TH1F *histPed = *mHistPedVecIter;
float meanPed = histPed->GetMean();
float rmsPed    = histPed->GetRMS();
histPed->GetXaxis()->SetRangeUser(meanPed-mPedCut*rmsPed,
                                  meanPed+mPedCut*rmsPed); // mPedCut = 3.0
```

(3) Get Mean/RMS as pedestal/RMS for each channel each time bin

```
meanPed     = histPed->GetMean();
rmsPed      = histPed->GetRMS();
```

The calculated values are written to a data file, and it's used to populate calibration Db table "istPedNoise". The output includes channel electronics ID, rdo ID, arm ID, apv ID, channel ID per chip, time bin index, calculated pedestal and RMS noise. The maker also uses the histogram method to calculate the common-mode noise for all chips over all time bins, including chip geometry ID, rdo ID, arm ID, apv ID, time bin index and calculated common-mode noise.

In addition, the maker will generate several QA plots. The calculated pedestal and RMS noise are shown in Fig. 9.

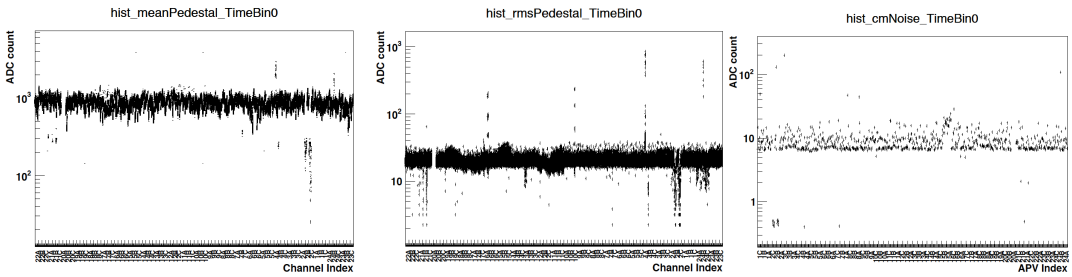

Fig. 9 Calculated pedestal (left) and RMS noise (middle) of all channels; Calculated common-mode noise (right) of all chips.

Currently, the gain calculation maker is not available yet, which will be located in the StIstCalibrationMaker directory. The gain for each channel is set to 1.0 and has been populated to offline DB.

7. QA maker (StIstQAMaker)

StIstQaMaker is designed to generate histograms and trees for fast QA on both IST raw hit and IST hit. The The Fig. 10 shows the hit global position and map selected from QA histograms (top-left: hit map; top-right: hit map of ladder ID vs. APV ID; bottom-left: global position in XY view; bottom-right: global position in Z-phi view).

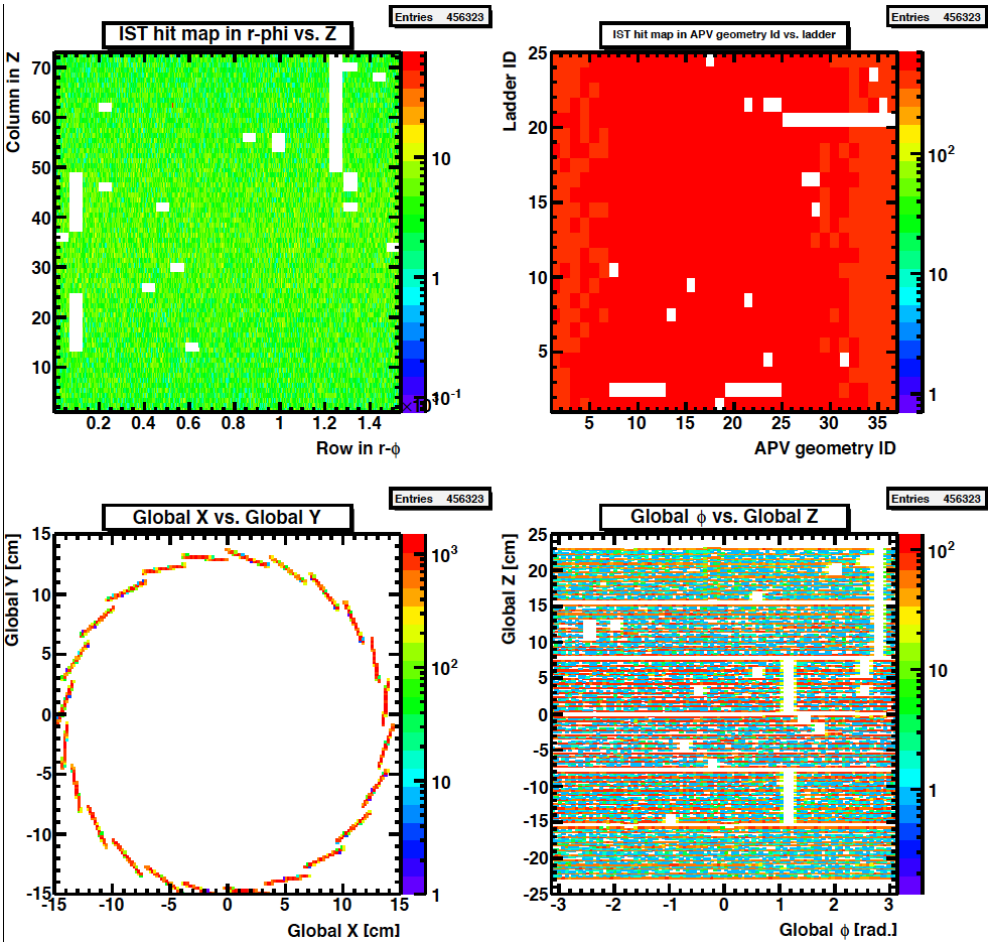

Fig. 10 IST hit global position

8. Fast simulation maker (StIstFastSimMaker)

The StIstFastSimMaker takes StMcIstHit as input, which is generated by StarSim output. In the fast simulator, the smearing effect is considered. The MC hits are packed as StIstHit format, and are stored into StIstHitCollection. The StiIstDetectorBuilder and StiIstHitLoader (located in StRoot/StiIst) are used for IST tracking. The Fig. 11 shows the IST hit global positions reconstructed by the fast simulator

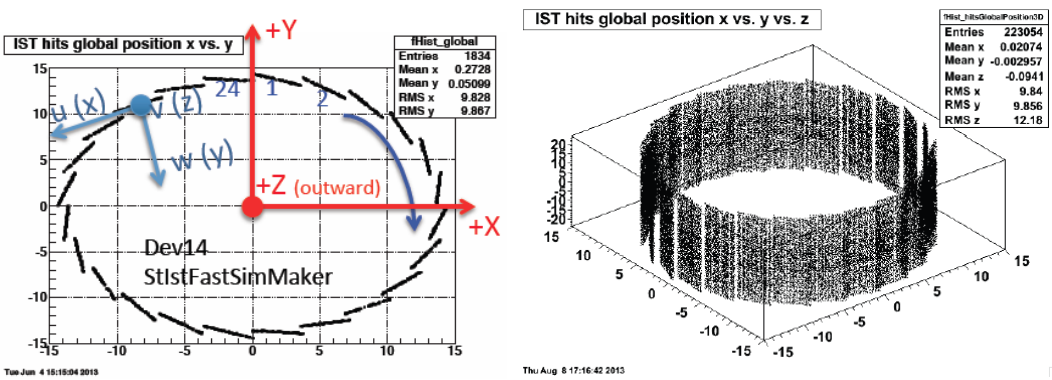

Fig. 11 IST fast simulation results (Left: global position in XY view; Right: global position in XYZ view)

9. Codes installation and run instruction

The latest codes are at: $CVSROOT/offline/hft/StRoot

```
starver dev
mkdir work_dir
cd work_dir
cvs co –r HEAD offline/hft

cd StRoot
ln -s ../offline/hft/StRoot/StIstCalibrationMaker
ln -s ../offline/hft/StRoot/StIstClusterMaker
ln -s ../offline/hft/StRoot/StIstFastSimMaker
ln -s ../offline/hft/StRoot/StIstHitMaker
ln -s ../offline/hft/StRoot/StIstQAMaker
ln -s ../offline/hft/StRoot/StIstRawHitMaker

cd ..
cons
```

The StIstDbMaker and StIstUtil packages have been written to STAR library, and all IST related codes in StEvent have been moved to the StEvent too. All IST geometry/calibration Db tables have been populated to the STAR DB. To run IST offline/calibration chain, we do not need the local StarDb.

(1) To run IST offline reconstruction chian:

```
cp /star/u/ypwang/disk01/offline_hft/runBFC.sh work_dir/
sh runBFC.sh
```

(2) To run IST calibration chain:

```
cp /star/u/ypwang/disk01/offline_hft/runBFC_calib.sh work_dir/
cp /star/u/ypwang/disk01/offline_hft/bfc_calib.C work_dir/
sh runBFC_calib.sh
```

Appendix 1:

The offline software list:

StRoot/StEvent/

- StIstHit.h/cxx
- StIstHitCollection.h/cxx
- StIstLadderHitCollection.h/cxx
- StIstSensorHitCollection.h/cxx

StRoot/StIstUtil/

- StIstConsts.h
- StIstRawHit.h/cxx
- StIstRawHitCollection.h/cxx
- StIstCluster.h/cxx
- StIstClusterCollection.h/cxx
- StIstCollection.h/cxx
- StIstDigiHit.h/cxx

StRoot/StIstRawHitMaker/

- StIstRawHitMaker.h/cxx

StRoot/StIstClusterMaker

- StIstClusterMaker.h/cxx
- StIstIClusterAlgo.h.h/cxx
- StIstSimpleClusterAlgo.h/cxx
- StIstScanClusterAlgo.h/cxx

StRoot/StIstHitMaker

- StIstHitMaker.h/cxx

StRoot/StIstDbMaker

- StIstDbMaker.h/cxx

StRoot/StIstCalibrationMaker

- StIstCalibrationMaker.h/cxx

StRoot/StIstQAMaker

- StIstQAMaker.h/cxx

StRoot/StIstFastSimMaker

- StIstFastSimMaker.h/cxx

StRoot/StEvent/

- StIstContainer.h: lines 162, 220
- StIstContainer.cxx: lines 146, 204
- StEvent.h: lines 206, 268-269, 339
- StEvent.cxx: lines 223, 783-797, 1247-1251, 1367, 1393
- StEventType.h: lines 278-281
- StEventClusterHints.cxx: lines 165, 189
- StEnumerations.h: //lines 533-542

StRoot/StEventUtilities/

- StEventHitIter.cxx: lines 20, 450-525, 750
- StuFixTopoMap.cxx: lines 126-131

StRoot/StBFChain/

- BigFullChain.h: following lines are added (1349-1356)

```
{"istDb"      ,"istDb","","tpcDb","StIstDbMaker","StIstDbMaker","Load
and run IstDbMaker",        kFALSE},
{"istFastSim","","","StMcEvent,StEvent",
"StIstFastSimMaker","StIstFastSimMaker","FastIstSimulator",kFALSE},
{"istUtil"     ,"","","","","StIstUtil","Ist Utilities",kFALSE},
{"istRaw"      ,"","","istUtil,istDb","StIstRawHitMaker","StIstRawHit
Maker","Ist Raw Hit Maker",kFALSE},
{"istCluster"      ,"","","istUtil","StIstClusterMaker","StIstCluster
Maker","Ist Cluster Maker",kFALSE},
{"istHit"      ,"","","istUtil,event,istDb","StIstHitMaker","StIstHit
Maker","Ist Hit Maker",kFALSE},
{"istQA"    ,"","","istUtil,StEvent"        ,"StIstQaMaker","StIstQaMa
ker","Example of Ist QA",kFALSE},
```