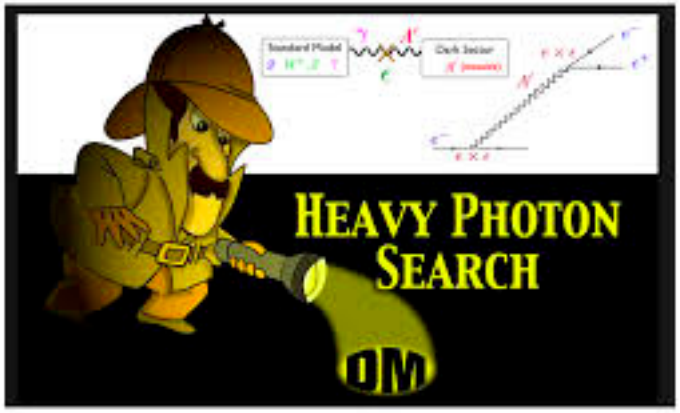
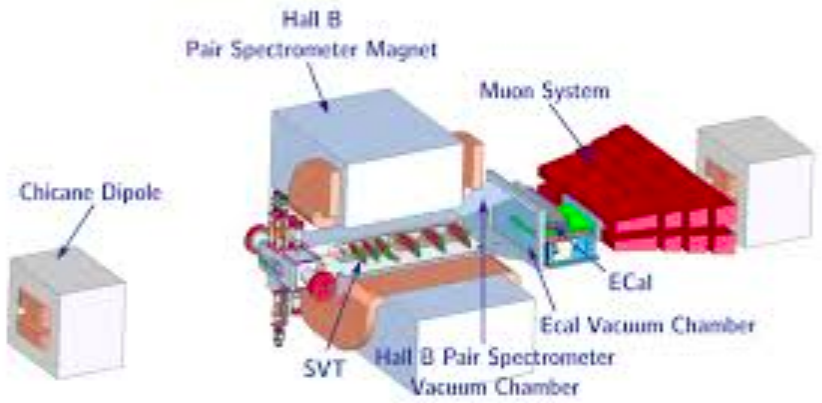




Status of HPS MC

Tongtong Cao (UNH)

HPS Collaboration Meeting
Nov. 18 – 19, 2019

Jefferson Lab

University of New Hampshire

Outline

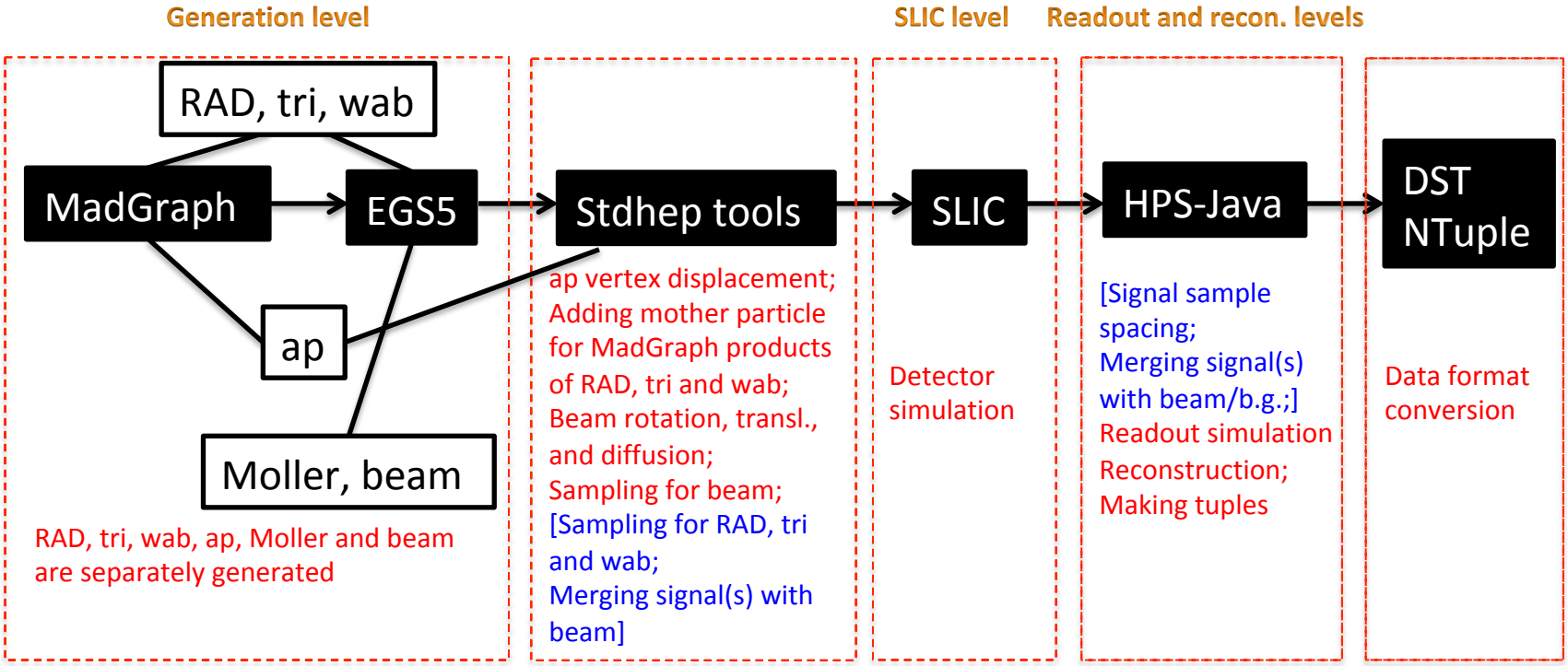
- What MC data for HPS
- Procedure of MC data production
- How to Make Plan for MC Data
- Status of HPS 2016 MC
  - Setup
  - Issues for old data
  - New data production and discussion
- Discussion for HPS 2019 MC
- Summary

What MC Data for HPS

$$S_{bin,zCut} = N_{bin} \frac{N_{rad}}{N_{tot}} \frac{3\pi\epsilon^2}{2N_{eff}\alpha} \frac{m_{A'}}{\delta m_{A'}} \epsilon_{bin} \int_{zCut}^{zMax} \frac{e^{-(z-z_{tgt})/\gamma c\tau}}{\gamma c\tau} \epsilon_{vtx}(z, m_{A'}) dz$$

- Radiative fraction as a function of invariant mass of e^+e^- pair
- Determination of bin width ($\delta m_{A'}$ = factor * A' mass resolution)
  - A' mass resolution as a function of A' mass; scaled according to difference between Moller mass resolution between MC and experimental Moller data
  - Determination of factor
- Vertex reconstruction efficiency as a function of z and A' mass
- $zCut$ as a function of invariant mass of e^+e^- pair (MC data cannot reach requirement of statistics, so we use experimental data)
- Others, such as analyses for the event selection
- Note: Parameters extracted by analyzing MC data should be updated as conditions of the event selection are updated.

Procedure of MC Data Production



- Signal(s) and beam/background could be merged before or after SLIC level. According to where to merge, we classify MC data as 3 types:
  1. Merging before SLIC: WBT (unbiased beam), etc
  2. Merging after SLIC: ap-beam, RAD-beam, tritrig-beam, wab-beam, etc
  3. Merging before SLIC to build background, and then merging after SLIC for signal(s) and background: ap-WBT, etc
- Difference for merging before and after SLIC:
  - Merging before SLIC: Signal is sampled by Poisson according to integrated cross section, so normalization of final data sample should not consider integrated cross section
  - Merging after SLIC: Signal sample is spaced with fixed interval, so normalization of final data sample should consider integrated cross section

How to Make Plan for MC Data

Radiative fraction: N_{rad}/N_{tot}

- N_{rad} and N_{tot}:
  - N_{rad}: # of reserved RAD events in a given bin; but need to consider beam effects on RAD at each MC processing level and physics analysis level
  - N_{tot}: # of all events in a given bin, including trident, wab and background events
- What MC data samples:
  - To get N_{rad}:
    - Data sample for RAD-beam -> # of RAD + background events; but background cannot be removed from final data sample
    - Data sample for pure RAD -> # of reserved RAD; but effects of beam are ignored
  - To get N_{tot}: data samples of tri-beam and wab-beam
- What size of data samples:
  1. Estimate requirement of statistics for the analysis
  2. Determine where to merge, before SLIC or after SLIC
  3. Estimate sizes of data samples for signals by MadGraph and beam by EGS5 according to rates of original input to final output (for after SLIC) or rates per ms (for before SLIC)
- Make a chain for MC production

Setup of 2016 MC

- Basic parameters:
  - Beam energy (E_{beam}): 2.3 GeV
  - # of electrons per beam bunch (ne): 2500 (200 nA current)
  - Target thickness (dz): 0.0004062 cm
  - Target density: 6.306E-2 $cm^{-1}barn^{-1}$
  - ap mass: 50 55 60 65 70 75 80 85 90 95 100 125 150 175
- Beam rotation, translation and diffusion
  - Rotation: θ_x = -0.33 mrad, θ_y = 30.17 mrad, θ_z = 261.8 mrad (15 deg)
  - Translation: x = -0.224 mm, y = -0.08 mm, z = -4.3 mm
  - Diffusion: σ_x = 0.125 mm, σ_y = 0.030 mm
- Cut parameters in MadGraph and EGS5
  - MadGraph:
    - ap: $\theta_y > 0$, $E > 0$ for e^+ and e^-
    - RAD: θ_y > 5 mrad, $E >$ 50 MeV for e^+ and e^-
    - tritrig: θ_y > 5 mrad, $E >$ 100 MeV for e^+ and e^-
    - wab: θ_y > 5 mrad, $E >$ 400 MeV for all particles
  - EGS5:
    - All interactions: Get rid of photons with θ_y < 5 mrad and electrons with ($E < 0.005E_{beam}$) or ($E > 0.6E_{beam}$ && $\sqrt{\theta_x^2 + \theta_y^2} <$ 5 mrad)
    - Especially for bremsstrahlung: Get rid of phtons with θ_y > 5 mrad and E_{min} = 400 MeV to avoid overlapping with wab generated in MadGraph

Issues of Old Data

- Issue list:
  - In stdhep tools (c++ & fortran)
    - Memory leak
    - Beam sampling issue -> beam bunches are not completely independent
    - Function open_read() can not correctly return # of events of an opened file (Bug: Changes made to a parameter inside a function have no effect on the corresponding argument)
  - In job submission scripts (shell script & xml):
    - For some data samples, every 100 jobs repeatedly use the same input files for both signal and beam at SLIC level, which causes MC events are not independent
    - Trivial issues: Wrong directory for the DST command
  - Plenty of auger jobs were failed. However, jobs are submitted in sequence at each level, so failed jobs with random job numbers cause that MC processing chain is not well organized. We should find reasons of fails, fix issues, make scripts to extract failed-job numbers for re-submission
  - Statistics for some data samples are not enough
  - Background for some data samples is overkill, such as data samples of RAD-WBT, tritrig-WB and wab-BT for analysis of the radiative fraction
- Since all above reasons, most of old data samples are not available
- All found issues have been fixed for new data production

Issue for Beam Sampling in Stdhep Tools

```
while (true) {
        bool no_more_data = false;
        while (!read_next(istream)) {
            close_read(istream);
            if (optind<argc-1)
            {
                open_read(argv[optind++],istream);
            }
            else
            {
                no_more_data = true;
                break;
            }
        }
        if (no_more_data) break;

        vector<stdhep_entry> * read_event = new
vector<stdhep_entry>;
        read_stdhep(read_event);
        input_events.push_back(read_event);
    }
```

```
while (file_n<=max_output_files) {

sprintf(output_filename,"%s_%0*d.stdhep",output_basename,output
_filename_digits,file_n++);
            open_write(output_filename,ostream,output_n);
        for (int nevhep = 0; nevhep < output_n; nevhep++)
        {
            int n_merge = gsl_ran_poisson(r,poisson_mu);
            if (n_merge==0)
                add_filler_particle(&new_event);
            for (int i=0;i<n_merge;i++)
            {
                int random_index =
gsl_rng_uniform_int(r,input_events.size());

append_stdhep(&new_event,input_events[random_index]);
            }

            write_stdhep(&new_event,nevhep+1);
            write_file(ostream);
        }
        close_write(ostream);
    }
```

- All events in input files are pushed into a vector, which causes very large memory usage
- Events are randomly picked up from the event vector to build beam bunches, which causes that events in the vector may be repeatedly used, and further causes that produced beam bunches are not completely independent of each other

Contents in /u/group/hps/production/mc/EngRun2015Scripts/slicwab-beam-tri_2pt3_tilt.xml

Issue in Job Submission Scripts

```
<List name="num">1</List>
 <Variable name="num100" value="1"/>
 <ForEach list="num">
  <Job>
    <Input src="mss:/mss/hallb/hps/production/stdhep/beam_5mrad/${ebeam}/egsv6_${num100}.stdhep" dest="beam.stdhep" />
    <Input src="mss:/mss/hallb/hps/production/MG_alphaFix/lhe/tritrig_100to1/${ebeam}/tritrigv2_5mrad_${num100}.tar" dest="tri.tar" />
    <!--<Input src="file:/work/hallb/hps/mc_production/MG_alphaFix/tritrig/${ebeam}/tritrigv2_5mradAF_ESum1GeV_${num100}.lhe.gz"
dest="tri.lhe.gz" />-->
    <Input src="file:/work/hallb/hps/mc_production/MG_alphaFix/wab/${ebeam}/wabv3_400MeV_5mrad_${num100}.lhe.gz"
dest="wab.lhe.gz" />
```

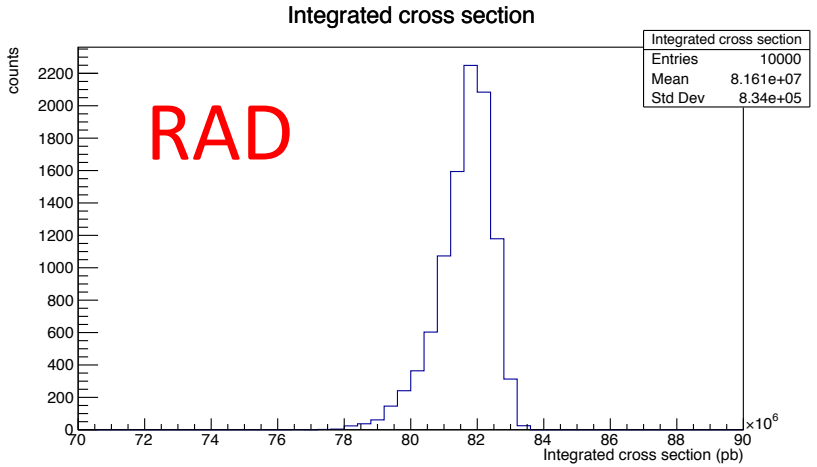
A shell script calls this xml file. In the shell script, 1 in <List> is replaced by 100 consecutive numbers by the sed command. So 100 consecutive jobs are submitted. But the jobs use the same input files, which causes events in final data samples are not independent of each other.

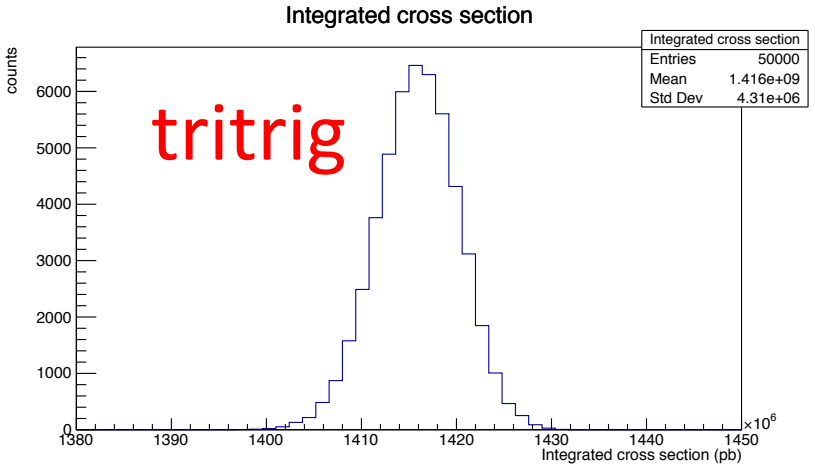
New Data Production

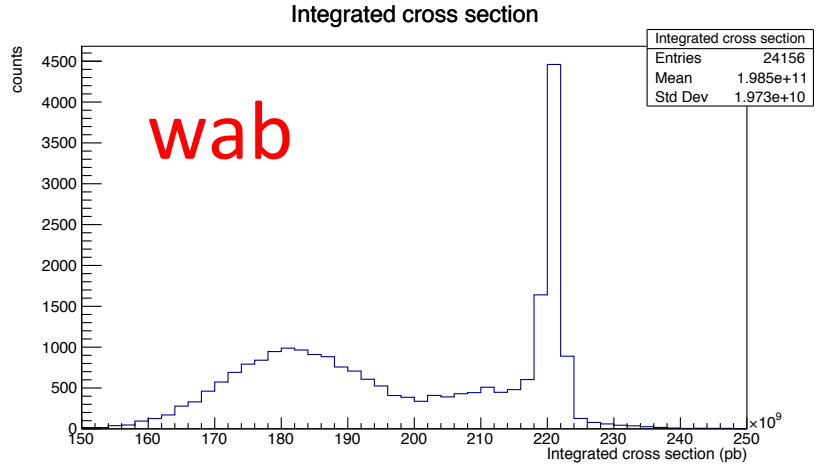
- Generation level:
  - MadGraph: /w/hallb-scifs17exp/general/hps/mc_production/MG_alphaFix/[ap : RAD] and /mss/hallb/hps/production/lhe/MG_alphaFix/[tritrig_100kFiles : wab_100kFiles]
    - ap: mass points (50 55 60 65 70 75 80 85 90 95 100 105 110 115 120 125 130 135 140 145 150 175); 200 files for each mass point, 100 files for prompt and 100 files for 10 mm displacement; # of events per file is not fixed, but larger than 10k
    - RAD: 10k files, and 10k events per file
    - tritrig: 50k files, and 10k events per file
    - wab: 100k files, and 10k events per file
  - EGS5: /w/mss/hallb/hps/production/stdhep/beam_5mrad
    - Beam: 10k files, 2500k bunches per file
- SLIC level: /mss/hallb/hps/production/BeamTilt/physrun2016/tpass2b/npt224npt08n4pt3_npt33/slic/[ap : RAD : tritrig : tritrig_100kFiles : wab : beam_2500kBunches]
  - ap, RAD, tritrig, wab: # of files are the same as samples at generation level, correspondingly
  - Beam: 9998 files; 2 jobs failed due to huge memory usage
- Reconstruction level with data format conversion:
  - Location:
    - .slcio: /mss/hallb/hps/production/PhysicsRun2016/pass4/npt224npt08n4pt3_npt33/recon
    - DST: /mss/hallb/hps/production/PhysicsRun2016/pass4/npt224npt08n4pt3_npt33/dst
    - tuple: /work/hallb/hps/mc_production/PhysicsRun2016/pass4/tuple/npt224npt08n4pt3_npt33
  - Subfolders:
    - ap-beam_2500kBunches: 200 files for each mass point; 100 files for prompt and 100 files for 10 mm displacement
    - RAD-beam_2500kBunches: 9998 files
    - RAD: 10k files
    - tritrig-beam_2500kBunches: 9998 files; 5 to 1 bundling for tritrig when application of spacing
    - wab-beam_2500kBunches: 9998 files; 10 to 1 bundling for wab when application of spacing
    - WBT: 50s unbiased beam in plan

Data from Madgraph at generation level

Integrated Cross Section







- Mean for integrated cross section
  - RAD: 81.61 ub
  - tri: 1.416 mb
  - wab: 0.1985 b
- Mean for # of events per beam bunch
  - RAD: 5.226E-6
  - tri: 9.068E-5
  - wab: 1.271E-2

Information is extracted from log files

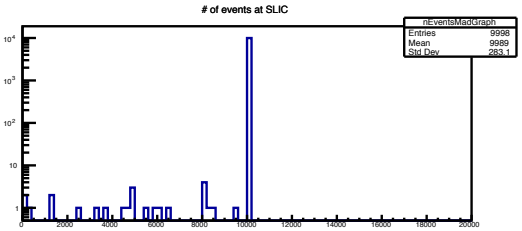
Evolution for # of Events per File

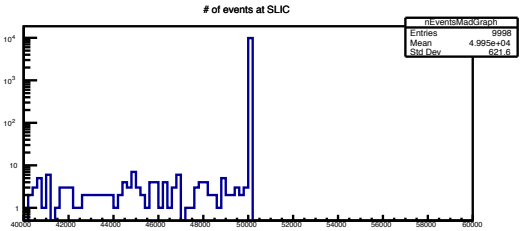
RAD-beam
10k original events per file from MadGraph

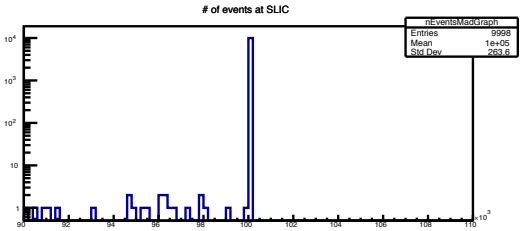
tritrig-beam
50k original events per file from MadGraph

wab-beam
100k original events per file from MadGraph

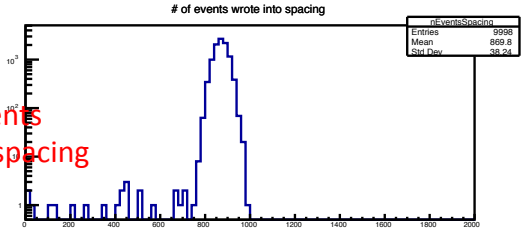
After SLIC

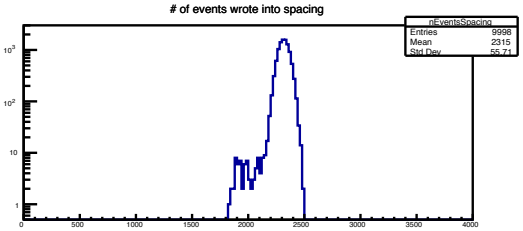


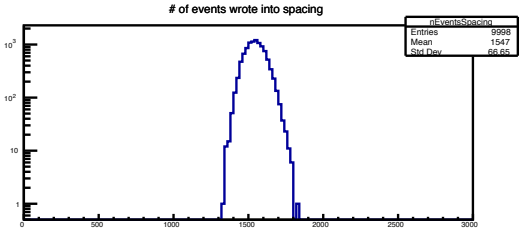




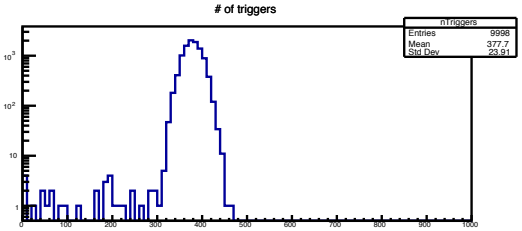
Available events written into spacing

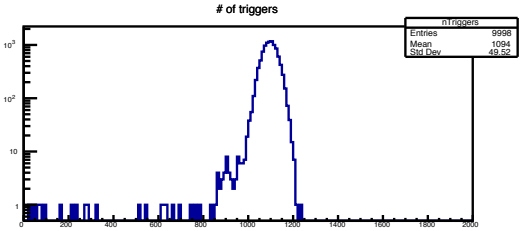


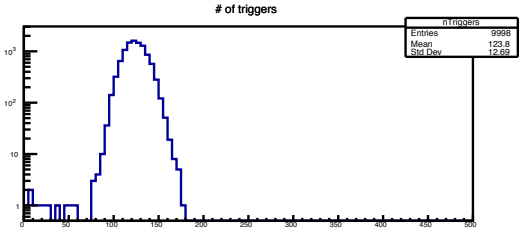




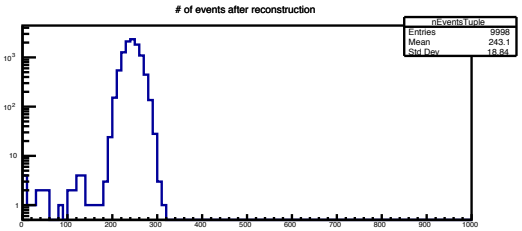
# of triggers

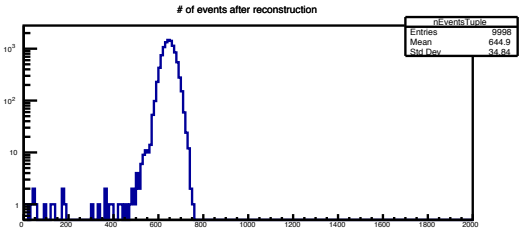


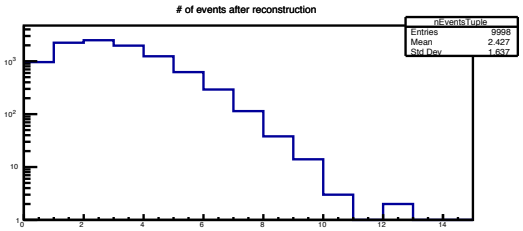




After Recon.







What to Learn from Evolution

- Check job processing so as to identify if a job has problem at each level and eliminate problematic files from final tuple/DST data samples; A list for problematic files in final data samples is made for each data sample; # of problematic files: 35/9998 for wab-beam, 146/9998 for tritrig-beam, and 34/9998 for wab-beam
- Calculate rates of original input to final output and rates per ms
  - For data sample without cross section (merging after SLIC), rates of original input to final output are extracted as:
    - RAD: ~243 output per 10k input
    - trigrig: ~545 output per 50k input
    - wab: ~2.43 output per 100k input
  - For data sample with cross section (merging before SLIC), rates per ms are estimated as:
    - RAD: ~0.0635 per ms
    - trigrig: ~0.4942 per ms
    - wab: ~0.1544 per ms
- Estimate statistics for data samples in plan: 50s WBT are estimated to include ~24710 tritrig events and ~7720 wab events

Discussion

- Mutual effects among signals: According to mean values for # of signal events, probability that a tri event and a wab event take place in the same beam bunch is ~1.15E-6, so the mutual effect is negligibly small. Integrated cross section of RAD is much smaller than tri and wab, so mutual effects between RAD and others is ignorable.
- Difference for merging before and after SLIC, i.e. mutual effects between signals and beam for detection processing at SLIC level: Effects should be slight; we can test if # of events in WBT data samples is consistent with estimation after the sample is produced.
- Hadron effects: Plan to produce two unbiased beam data samples, WBT and WBTH, for comparison
- Procedure of old Moller data will be double checked to determine if need to be re-produced

2019 MC

- Basic parameters:
  - Beam energy (E_{beam}): 4.56 GeV
  - # of electrons per beam bunch (ne): 150 nA current?
  - Target thickness (dz): 8 um?
  - Target density: 6.306E-2 $cm^{-1}barn^{-1}$?
  - ap mass: mass points ?
- Beam rotation, translation and diffusion
  - Rotation: ?
  - Translation: ?
  - Diffusion: ?
- Cut parameters in MadGraph and EGS5?
- Updates for readout simulation, reconstruction, drivers for tuple in HPS Java?
- Updates for NTuple and DST?
- More …

Summary

- For 2016 MC,
  - Issues for old data are discussed and fixed
  - New MC data were produced last week
  - More data sample production is in plan
- For 2019 MC, a lot of effects need to be contributed for data production

Backup Slides

Issues and solutions for stdhep tools

Memory Leak in stdhep_util.cpp

```
int read_stdhep(vector<stdhep_entry> *new_event)
{
     int offset = new_event->size();
    for (int i = 0;i<hepevt_.nhep;i++)
    {
        struct stdhep_entry *temp = new struct stdhep_entry;
        temp->isthep = hepevt_.isthep[i];
        temp->idhep = hepevt_.idhep[i];
        for (int j=0;j<2;j++) {
            temp->jmohep[j] = hepevt_.jmohep[i][j];
            if (temp->jmohep[j]!=0) temp->jmohep[j]+=offset;
        }
        for (int j=0;j<2;j++) {
            temp->jdahep[j] = hepevt_.jdahep[i][j];
            if (temp->jdahep[j]!=0) temp->jdahep[j]+=offset;
        }
        for (int j=0;j<5;j++) temp->phep[j] = hepevt_.phep[i][j];
        for (int j=0;j<4;j++) temp->vhep[j] = hepevt_.vhep[i][j];
        new_event->push_back(*temp);
    }
     return hepevt_.nevhep;
}
```

```
void write_stdhep(vector<stdhep_entry> *new_event, int nevhep)
{
     hepevt_.nhep = new_event->size();
     hepevt_.nevhep = nevhep;
     //vector<stdhep_entry>::iterator it;
    for (int i = 0; i<new_event->size(); i++)
    {
        struct stdhep_entry temp = new_event->at(i);
        hepevt_.isthep[i] = temp.isthep;
        hepevt_.idhep[i] = temp.idhep;
        for (int j=0;j<2;j++) hepevt_.jmohep[i][j] =
temp.jmohep[j];
        for (int j=0;j<2;j++) hepevt_.jdahep[i][j] =
temp.jdahep[j];
        for (int j=0;j<5;j++) hepevt_.phep[i][j] = temp.phep[j];
        for (int j=0;j<4;j++) hepevt_.vhep[i][j] = temp.vhep[j];
    }
  has_hepev4 = false;
     new_event->clear();
}
```

Issue: objects are created by “new” and pushed into a vector in read function, but memory allocated for objects can not be released by vector::clear() in write functions.

Solution: To not touch other source files, we use another vector, which is static global so as not to affect source codes of the package, to collect all points defined in read function, so that memory can be released in write functions

Return value for open_read() in stdhep_util.cpp

```
int open_read(char *filename, int istream, int n_events)
{
    printf("Reading from %s; expecting %d
events\n",filename,n_events);
    if ( xdr_init_done )
        StdHepXdrReadOpen(filename,n_events,istream);
    else
    {
        StdHepXdrReadInit(filename,n_events,istream);
        xdr_init_done = true;
    }
     return n_events;
}
```

Issue: The function is supposed to return # of events after opening a file, but actually cannot since changes made to a parameter inside a function have no effect on the corresponding argument. However, # of events per file needs to be used in other source codes, like in random_sample.cc.

Solution: Build two new functions StdHepXdrReadInitNTries(char *filename, int *ntries, int ist) and StdHepXdrReadOpenNTries(char *filename, int *ntries, int ist) in stdhep/src/ stdhep-5-06-01/src/stdhep to replace StdHepXdrReadOpen(filename,n_events,istream) and StdHepXdrReadInit(filename,n_events,istream) in open_read() so as to pass parameter's value by pointer

Issues in random_sample.cc

```
while (true) {
        bool no_more_data = false;
        while (!read_next(istream)){
            close_read(istream);
            if (optind<argc-1)
            {
                open_read(argv[optind++],istream);
            }
            else
            {
                no_more_data = true;
                break;
            }
        }
        if (no_more_data) break;

        vector<stdhep_entry> * read_event = new
vector<stdhep_entry>;
        read_stdhep(read_event);
        input_events.push_back(read_event);
    }
```

```
while (file_n<=max_output_files) {

sprintf(output_filename,"%s_%0*d.stdhep",output_basename,output
_filename_digits,file_n++);
          open_write(output_filename,ostream,output_n);
        for (int nevhep = 0; nevhep < output_n; nevhep++)
        {
            int n_merge = gsl_ran_poisson(r,poisson_mu);
            if (n_merge==0)
                add_filler_particle(&new_event);
            for (int i=0;i<n_merge;i++)
            {
                int random_index =
gsl_rng_uniform_int(r,input_events.size());

append_stdhep(&new_event,input_events[random_index]);
            }

            write_stdhep(&new_event,nevhep+1);
            write_file(ostream);
        }
        close_write(ostream);
    }
```

Issue: 1) All events in input files are pushed into a vector, which causes very large memory usage 2) Events for a beam bunch are randomly picked up from the vector, which causes that events in the vector may be repeatedly picked up, and further output beam bunches are not completely independent of each other

Solutions to Issues of random_sample.cc

- Solution 1: Develop a new code called as random_sample_usingInputEventsInOrder.cc; In the code, vector is not necessary to be used and events in input files are used in order to build beam bunches
- Currently, we have produced 10000 beam files with 2500k bunches per file by EGS5. These files are enough for current MC production plan, and are not necessary to be shared to produce MC files. So currently solution 1 works well. But if we need larger data sample in the future and beam files need to be shared, we'd better randomly pick up events from input files.
- Solution 2: As suggested by Maurik, a proper way to sample random events from a vector is to create a list of indices: vector<int> event_list; fill it with 0….N, and then randomize the vector. We randomize the vector with: std::shuffle(event_list.begin(),event_list.end(),random_gen); This ensures that the events are used in random order, and are used only once. Besides, we can cache events in batches of some reasonably large number, and then draw from that batch in random fashion so as to avoid too large memory usage. Note: to use std::shuffle, must be enabled with the -std=c++11 or -std=gnu++11 compiler options