Issues and Solutions for stdhep tools

Tongtong Cao

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 functions, 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 other source codes of the package, to collect all pointers used 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 the updated code 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

Introduction to Old random_sample.cc

- We use EGS5 to build electrons for beam bunches. Suppose that we produce N (= n * m) electrons from EGS5, where n is # of required bunches, m is # of electrons per bunch
- Then, we use random_sample.cc to build n beam bunches. # of electrons for each bunch is random by Poisson with mean of m.
- Issue 1: We can not guarantee that there are enough events to build n bunches since # of electrons for each bunch is random
- Solution to issue 1: Produce a little bit more events by EGS5 as $N^{'} > N + 5\sqrt{N}$
- Please use option −m to set “mean” parameter for Poisson, otherwise, it will be calculated as $N^{'}/n$

Issues 2 & 3 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 2: All events in input files are pushed into a vector, which causes very large memory usage, depending on # of events from input file(s) produced by EGS5.
- Issue 3: 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

Discussion for “Dependent” Issue

- Probability that a electron is not used in the whole processing of random beam sample:

$$\lim_{N\to\infty}(1-\frac{1}{N})^N = \frac{1}{e} \approx 0.37$$

- About 37% electrons produced by EGS5 are not used, which is a big waste.
- In other words, roughly 37% electrons are repeatedly used.
- It may not cause big issue for our analysis by MC data, but we can completely avoid the issue by improving codes.

Solution 1: New codes -

random_sample_usingInputEventsInOrder.cc

- We have produced enough electrons by EGS5 to build beam bunches, so it is not necessary to randomly pick up events from the whole input data sample.
- In the new code, electrons are picked up in order from input data to build beam bunches.
- The code is applied by new MC production in JLab. It completely avoids large memory usage and dependence issue. No waste for electrons produced by EGS5.

Solution 2: Updates of random_sample.cc

- Input events are cached into vectors one by one, where size of vectors is set by option −b (default: 1000000); then use std::shuffle to rearrange indices of elements in vector and pick up electrons in order of rearranged indices to build bunches.
- This code avoids too large memory usage, and can control memory usage by option −b. Additionally, it provides us a way to build multi beam bunch samples using the same input data sample by EGS5 if required. After all, cost for beam data by EGS5 is quite large if we need a large MC sample with beam background. No waste for electrons produced by EGS5.