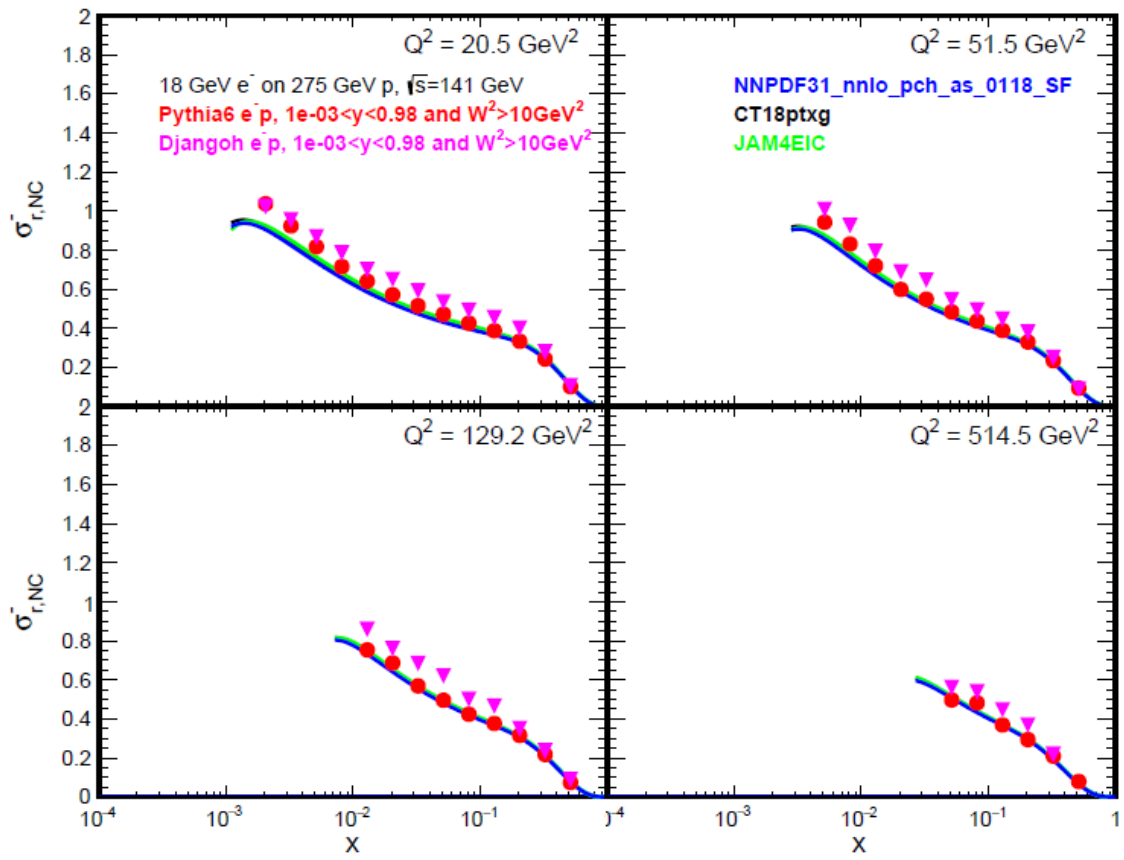
Random Number Seeding in *PythiaeRHIC*, *DJANGOH*, and *BeAGLE*

Barak Schmookler (SBU Postdoc)
Dmitriy Kim (SBU Undergraduate)
Daniel Waxman (SBU Undergraduate)

Motivation

- For many of the EIC Yellow Report (and future) simulation work, it is necessary to generate large numbers of events
- For *PythiaeRHIC* and *DJANGOH* it takes about 0.5-1 hour to generated 1 million minimum bias events on the BNL (RCF) machines
- For *BeAGLE*, it takes about 30 hours to generate 1 million minimum bias events for an electron-heavy ion reaction
- So it is obviously required to run multiple simulation jobs simultaneously



Simple Job Submission on RCF

Submission Script

```
Universe        = vanilla
Notification    = Never
Executable      = run_minbias.sh
GetEnv          = True
Input           = /dev/null
Arguments       = "$(Process)"
Output          = jobout/minbias_job.out.$(Process)
Error           = jobout/minbias_job.err.$(Process)
Log             = jobout/minbias_job.log.$(Process)
Queue 15
```

Shell Script

```
#!/usr/bin/bash

if [ -z "$1" ]
then
        echo "No job number set."
        echo "Please run as ./run_minbias.sh jobnumber"
        echo "Exiting..."
        exit 1
fi

echo "-----------------------------------"
echo "Running PYTHIA Simulation for ep Collider!!!"
echo "-----------------------------------"
echo "Performing Job $1"
echo "..."
echo ""

VAR1=$1
PythiaCard="s/ep_minbias.out/ep_minbias_${VAR1}.out/g"

sed "${PythiaCard}" ./infiles/other_studies/10_100/ep_minbias.inp > input_minbias_$1.inp

echo "Running PYTHIA..."
pythiaeRHIC < input_minbias_$1.inp > logfiles/other_studies/10_100/ep_minbias_$1.log

echo "Completed Simulation!!!"
echo ""

echo "Making Output ROOT File..."
root -l -b -q "make_tree.C(\"ep_minbias_$1.out\")"
echo "Done!!!"
echo ""

echo "Cleaning up..."
mv -vf ./outfiles/ep_minbias_$1.out ./outfiles/other_studies/10_100
mv -v ./outfiles/ep_minbias_$1.root ./outfiles/other_studies/10_100
rm -vf input_minbias_$1.inp
echo "Done!!!"
echo ""
```

How do the random seeds look for 10 jobs?

PythiaeRHIC

ep_minbias_0.log: SEED = 1996269241
ep_minbias_1.log: SEED = 469140822
ep_minbias_2.log: SEED = 884076946
ep_minbias_3.log: SEED = 923366337
ep_minbias_4.log: SEED = 374194774
ep_minbias_5.log: SEED = 574675150
ep_minbias_6.log: SEED = 113151324
ep_minbias_7.log: SEED = 1712347962
ep_minbias_8.log: SEED = 1483163476
ep_minbias_9.log: SEED = 1838488092

DJANGOH or BeAGLE

eAu_0.log: SEED = 2411
eAu_1.log: SEED = 2421
eAu_2.log: SEED = 2296
eAu_3.log: SEED = 2296
eAu_4.log: SEED = 2421
eAu_5.log: SEED = 2136
eAu_6.log: SEED = 2321
eAu_7.log: SEED = 2136
eAu_8.log: SEED = 2391
eAu_9.log: SEED = 2391

How do the random seeds look for 10 jobs?

PythiaeRHIC

ep_minbias_0.log: SEED = 1996269241
ep_minbias_1.log: SEED = 469140822
ep_minbias_2.log: SEED = 884076946
ep_minbias_3.log: SEED = 923366337
ep_minbias_4.log: SEED = 374194774
ep_minbias_5.log: SEED = 574675150
ep_minbias_6.log: SEED = 113151324
ep_minbias_7.log: SEED = 1712347962
ep_minbias_8.log: SEED = 1483163476
ep_minbias_9.log: SEED = 1838488092

DJANGOH or BeAGLE

eAu_0.log: SEED = 2411
eAu_1.log: SEED = 2421
eAu_2.log: SEED = 2296
eAu_3.log: SEED = 2296
eAu_4.log: SEED = 2421
eAu_5.log: SEED = 2136
eAu_6.log: SEED = 2321
eAu_7.log: SEED = 2136
eAu_8.log: SEED = 2391
eAu_9.log: SEED = 2391

Simplest fix is to delay each job based on job number…

Add this to the shell script before running the generator:

```
VAR1=$1
VAR2=$((2 * ${VAR1}))
echo "Sleeping for ${VAR2} Seconds"
sleep ${VAR2}
```

Simplest fix is to delay each job based on job number… but this still does not work perfectly

Add this to the shell script before running the generator:

```
VAR1=$1
VAR2=$((2 * ${VAR1}))
echo "Sleeping for ${VAR2} Seconds"
sleep ${VAR2}
```

```
eAu_0.log: SEED =       2151
eAu_1.log: SEED =       2156
eAu_2.log: SEED =       2291
eAu_3.log: SEED =       2256
eAu_4.log: SEED =       2281
eAu_5.log: SEED =       2331
eAu_6.log: SEED =       2346
eAu_7.log: SEED =       2381
eAu_8.log: SEED =       2391
eAu_9.log: SEED =       2156
```

How is the random seed set in *PythiaeRHIC*?

The random number seed is set in the C++ wrapper script *UsingCardPythiaMain.cpp* (https://gitlab.com/eic/mceg/PYTHIA-RAD-CORR/-/blob/master/src/drivers/UsingCardPythiaMain.cpp) :

```
// Random seed. Note that this is NOT safe for batch jobs.
// better idea would be to get a seed from /dev/urandom on UNIX type machines.
// time_t initseed = time(0);
// initseed=42;
// Better c++11 solution:
std::random_device rd;
int initseed = rd();
pythia->SetMRPY(1, initseed);
cout << " SEED = " << initseed << endl;
```

It seems like initially the time was used to set the seed

But the issue for batch jobs was recognized and now the machine entropy (from /dev/urandom) is used to set the seed

How is the random seed set in *DJANGOH* and *BeAGLE*?

The time is used to set the seed (in *djangoh_h.f* and *dpm_pythia.f*):

```
call idate(today)    ! today(1)=day, (2)=month, (3)=year
call itime(now)      ! now(1)=hour, (2)=minute, (3)=second
initseed = today(1)+10*today(2)+today(3)+now(1)+5*now(3)
```

Fix for *BeAGLE* – directly seed in input file based on output of /dev/urandom file

```
#!/usr/bin/bash

if [ -z "$1" ]
then
        echo "No job number set."
       echo "Please run as ./run_eAu.sh jobnumber"
        echo "Exiting..."
        exit 1
fi

echo "-----------------------------------"
echo "Running BeAGLE Simulation for eAu Collider!!!"
echo "-----------------------------------"
echo "Performing Job $1"
echo "..."
echo ""

VAR1=$1
PythiaCard="s/eAu.txt/eAu_${VAR1}.txt/g"
BeAGLECard="s/S3ALL002/InpAu_${VAR1}/g"

sed "${PythiaCard}" S3ALL002 > InpAu_$1
sed "${BeAGLECard}" ./inputFiles/eAu.inp > ./inputFiles/eAu_$1.inp

SEED=`od -vAn -N2 -tu2 < /dev/urandom`
echo "The Random SEED is ${SEED// /}"
echo ""
sed -i "s/1234567/${SEED// /}/g" InpAu_$1
```

```
echo "Running BeAGLE..."
$BEAGLESYS/BeAGLE < inputFiles/eAu_$1.inp > logs/eAu_$1.log

echo "Completed Simulation!!!"
echo ""

echo "Making Output ROOT File..."
root -l -b -q "make_tree.C(\"eAu_$1.txt\")"
echo "Done!!!"
echo ""

echo "Cleaning up..."
rm -vf ./outForPythiaMode/eAu_$1.txt
rm -vf InpAu_$1
rm -vf inputFiles/eAu_$1.inp
echo "Done!!!"
echo ""
```

Documented here:
https://wiki.bnl.gov/eic/index.php/Simulations#High-Statistics_BeAGLE_Simulation

BeAGLE output seeds from 10 runs after fix

eAu_0.log: SEED = 14473
eAu_1.log: SEED = 30550
eAu_2.log: SEED = 55545
eAu_3.log: SEED = 4022
eAu_4.log: SEED = 28238
eAu_5.log: SEED = 36662
eAu_6.log: SEED = 12502
eAu_7.log: SEED = 57292
eAu_8.log: SEED = 29721
eAu_9.log: SEED = 37451

Fix for *DJANGOH* – read in random numbers from an input text file

- On the EIC DJANGOH wiki page (https://wiki.bnl.gov/eic/index.php/DJANGOH), the example input files have the **RNDM-SEEDS** parameter set to **-1 -1** . When this default parameter is used, the time sets the random seed as discussed above, and the seed is passed into the subroutines in **gmc_random.f** are used for random number generation.
- If, however, we set the **RNDM-SEEDS** parameter to **1 1**, then the generator uses a set of routines from the paper

  George Marsaglia, Arif Zaman, Wai Wan Tsang, Toward a universal random number generator, Statistics & Probability Letters, Volume 9, Issue 1,1990, Pages 35-39

  which require a set of numbers from an input text file.

Fix for *DJANGOH* – read in random numbers from an input text file

- According to the paper, the generator requires 97 24-bit fractions (as well as some other parameters, which can be kept constant) to be taken from a text file.
- The following Python script can be used to generate the text file prior to running the *DJANGOH* simulation. It should be executed in the shell script before running the generator. (The Python random module should use /dev/urandom to set the its own seed.)

```
#!/bin/python
import random

def get_U():
        return sum(2**-i * random.getrandbits(1) for i in range(1, 25))

C = 362436.0 / 16777216.0
CD = 7654321.0 / 16777216.0
CM = 16777213.0 / 16777216.0
IP = 97
JP = 33
with open('fort.8', 'w') as f:
        f.write(' '.join([str(get_U()) for i in range(97)]))
        f.write('\n')
        f.write('{} {} {} {} {}'.format(C, CD, CM, IP, JP))
```

Potentially better long-term solution – use /dev/urandom directly in the Fortran generators

Can probably do something like this:

```
integer :: un, istat
open(newunit=un, file="/dev/urandom", access="stream", &
        form="unformatted", action="read", status="old", iostat=istat)
if (istat == 0) then
            read(un) seed
            close(un)
else
```

See https://gcc.gnu.org/onlinedocs/gcc-4.8.5/gfortran/RANDOM_005fSEED.html