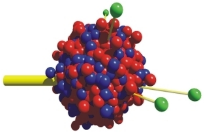
The GiBUU transport tutorial (part 2)

Janus Weil

FIAS, Frankfurt

HADES Winter School
Kloster Höchst, Feb. 2014



GiBUU

The Giessen Boltzmann-Uehling-Uhlenbeck Project

GiBUU: the code

- written in Fortran (mixture of dialects: F77/F90/F2003)
- Fortran is an 'old' language, but has been updated to include modern computing concepts (modularization, object orientation, ...)
- GiBUU code is reasonably modern (most parts in F90+)
- semi-automatic documentation (via RoboDoc)
- $\sim$ 100.000 LOC (rough order of magnitude)
- developed by a collaboration of several people (at Giessen and Frankfurt)
- managed through svn repository (version-control system)
- development branch (internal) and public releases (current: 1.6)
- warning: research code, work in progress ...

People

people currently involved with GiBUU:

- Ulrich Mosel (JLU Giessen, emeritus professor)
- Kai Gallmeister (Uni Frankfurt)
- Alexei Larionov (FIAS)
- Theo Gaitanos (JLU Giessen)
- J.W. (FIAS)

plus $\sim$ 100 users all around the world

The GiBUU website

- `https://gibuu.hepforge.org`
- central place for all information on GiBUU
- based on a wiki system ('trac')
- contains lots of information about the model and code
- documentation of input parameters, output files, etc
- source code viewer for svn repository
- timeline of news & changes

Technical prerequisites

- GiBUU runs on Linux, Mac, Windows
- Linux is preferred platform and will be used in this tutorial
- needed software tools:
  - subversion (for code checkout)
  - GNU make (stears the build process)
  - a Fortran compiler (e.g. gfortran 4.4+)
  - perl, libbz2

Getting the code

- … via check-out from svn repository!
- 1. create a new directory

```
mkdir GiBUU; cd GiBUU
```

- 2. check out the code

```
svn co --username hades
https://gibuu.hepforge.org/svn/releases/release1.6
./release1.6
```

- 3. check out the input files

```
svn co --username hades
https://gibuu.hepforge.org/svn/releases/buuinput1.6
./buuinput
```

- That's it: Now you have everything you need to run a GiBUU simulation!

Compiling the code

- basically: go to directory 'release1.6' and type '`make`' (and that's it!)
- takes about 2 min on my Notebook (using one core)
- alternative invocations:
  - parallel make (on multiple cores):
    `make -j4`
  - choosing a particular compiler:
    `make FORT=gfortran-4.8`
  - optimization (better performance but no debugging):
    `make MODE=opt3`
  - static linking:
    `make STATIC=1`
  - re-compile everything:
    `make renew`

EXCURSION: UPDATING THE SVN REPOSITORY

- from time to time, there will be changes in the code (bugfixes, new features, ...)
- then you should update your local copy of the code
- svn takes care of this: checks your version against the newest version on the server, downloads the differences and applies them
- one simple command (in 'release1.6' directory):
  ```
  svn update
  ```
- output shows which files are being updated, and to which revision
- after updating, you need to recompile:
  ```
  make
  ```

Running the code

- after successful compilation, the exectuable 'GiBUU.x' is located in the subdirectory 'objects/'
- simply run the executable with input and output files: ./GiBUU.x < input.job > log.txt
- either run it 'in-tree' (e.g. in the testrun directory: cd testrun; ./GiBUU.x), or copy it to some other place (recommended, since several output files are produced)
- the output file 'log.txt' contains a log of GiBUU control & debug messages, physics output will be written to other files

INPUT PARAMETERS(1)

- all input parameters are provided to GiBUU in form of a 'jobcard' (plain text file which contains input in special format)
- several example jobcards provided in subdirectory 'testrun/jobcards'
- format: jobcard contains several 'namelists', which consists of a number of input switches:

```
&namelist1
  switch1 = value1    ! some comment
  switch2 = value2    ! another comment
/

&namelist2
  switch3 = value3
  switch4 = value4
/
```

- as usual in Fortran, capitalization (upper/lower case) does not matter

Input parameters (2)

- there are a lot of different input parameters
- most of them not relevant for beginners, but only for changing advanced parameters of the model (most have a reasonable default value)
- all namelists and switches documented on website:
  `https://gibuu.hepforge.org/Documentation1.6/code/robo_namelist.html`
- for simplicity, we start by constructing an input jobcard for elementary p+p collisions
- some relevant namelists:
  1. 'input' (basic setup)
  2. 'elementary' (initialization of elementary reaction)
  3. 'LesHouches' (producing particle output)
  4. 'elementary_analysis' (additional analysis output)

Basics: The namelist 'input'

this contains all the basic settings that need to be supplied

```
&input
  eventtype    = 0            ! type of reaction: elementary coll.
  numEnsembles = 100000        ! number of ensembles
  numTimeSteps = 1            ! number of time steps
  num_runs_SameEnergy = 1     ! number of runs per energy
  num_energies        = 1     ! number of energies
  length_real = 20            ! length of real particle vector
                              ! (max. particles per ens.)
  path_To_Input = '/some/path/to/buuinput'
/
```

'path_to_input' must point to the local path of the 'buuinput' directory

The namelist 'elementary'

... defines the two elementary scattering partners (any hadron is allowed)

```
&elementary
  particleId(1:2)      =  1, 1   ! particle IDs: 1=nucleon
  particleCharge(1:2)  =  1, 1   ! charges
  particleAnti(1:2)    =  F, F   ! antiparticles? (F=.false.)
  ekin_lab             =  3.5    ! kinetic energy in GeV
                                 ! of first part.
/
```

second particle is always at rest ('fixed target')

Namelists 'LesHouches' and 'elementary_analysis'

```
&LesHouches
  LesHouchesFinalParticles_Real = T    ! print out real part.
  LesHouchesFinalParticles_Pert = F    ! print out pert. part.
/
```

- generate particle output in Les Houches format
- we only need real particles here (again: T=.true., F=.false.)
- output is written to a file called 'LesHouches.Real.*.xml'

```
&elementary_analysis
  DoOutChannels     = T
/
```

- produce 'OutChannels.dat' file, which lists produced channels and corresponding cross sections

Analysis Strategies

1. direct 'on-line' analysis inside GiBUU
   - direct analysis of desired quantity during the sim.
   - no intermediate particle output
   - directly produce histograms etc
   - advantage: access to all internal information

2. output all particles/events in LesHouches format,
   read into ROOT for analysis
   - 'off-line' analysis
   - analysis procedure can be changed after sim. is finished
   - may produce large amount of data

Output format: Les Houches

- XML-like event format
- named after a town in France
- arXiv:hep-ph/0609017v1
- rough structure:

```
<LesHouchesEvents version="1.0">
<header>
  ...
</header>
<init>
  ...
</init>
<event>
  ...
</event>
... (any number of <event> blocks can follow) ...
```

<EVENT> BLOCK FORMAT

```
<event>
   3  0  0.   0.   0.   0.
  2212  0  0  0  0  0 -0.264  0.275  3.468  3.613  0.938  0. 9.
  2112  0  0  0  0  0  0.267 -0.052  0.138  0.986  0.938  0. 9.
   211  0  0  0  0  0 -0.003 -0.222  0.730  0.776  0.138  0. 9.
</event>
```

- first line: N = number of particles (plus a few boring zeros)
- then: N lines, representing one particle each
- columns: 1=ID code, 7-9=p_x, p_y, p_z, 10=E, 11=mass [GeV]
- ID code: according to PDG numbering scheme

Exercises

1. run a p+p collision at $E_{kin} = 3.5 GeV$
2. have a look at the produced files
   - log.txt
   - OutChannels.*.dat
   - LesHouches.Real.*.dat
3. how many pions do you get per event $(\pi^+/\pi^-/\pi^0)$?
4. plot the energy spectra of all $\pi^+/\pi^-/\pi^0$ mesons
5. plot the p_T spectrum of all $\pi^+/\pi^-/\pi^0$ mesons
6. plot the same spectra for Kaons
7. plot the p_T spectra for exclusive $\pi^+/\pi^-/\pi^0$ production