The new C++ based HIJING event generator: HIJING++

Sz.M. Harangozó^{1,2}, G-Y. Ma^{3}, G.G. Barnaföldi^{2}, P. Lévai^{2}, G. Papp^{1,†}, X-N. Wang^{3,4}, B-W. Zhang^{3}

^{1}Institute for Physics, Eötvös University, Budapest, Hungary, ^{2}MTA Wigner RCP, Budapest, Hungary, ^{3}IOPP, Central China Normal University, Wuhan, China, ^{4}Nuclear Science Division, LBNL, USA

†e-mail: `pg@elte.hu`

Abstract

The popular HIJING event generator is redesigned to match the compatibility with AliRoot and is rewritten to C++. We review here the design of the C++ interface and the connections to the PYTHIA 8 event generator. Furthermore, with the development new physics is also introduced, like the inclusion of new particle distribution functions (LHAPDF), the DGLAP evolution of the shadowing effect, different jet-quenching models.

Introduction

The original HIJING [1] (Heavy Ion Jet INteraction Generator) Monte Carlo model was developed by M. Gyulassy and X.-N. Wang with special emphasis on the role of minijets in pp, pA and AA reactions at collider energies in a wide range from 5 GeV to 2 TeV. The program itself is written in FORTRAN, and is based on the FORTRAN version of PYTHIA (currently v6), ARIADNE and the CERNLIB package PDFLIB.

The main features included in HIJING are

- Soft beam jets are modeled by diquark-quark strings with gluon kinks along the lines of the Lund FRITIOF and dual parton model (DPM). In addition, multiple low p_T exchanges among the end point constituents are included to model initial state interactions.
- Multiple minijet production with initial and final state radiation is included along the lines of the PYTHIA model in an eikonal formalism.
- Exact diffuse nuclear geometry is used to calculate the impact parameter dependence of the number of inelastic processes.
- An impact-parameter-dependent parton structure function is introduced to study the sensitivity of observables to nuclear shadowing, especially of the gluon structure functions.
- A model for jet quenching is included to enable the study of the dependence of moderate and high p_T observables on an assumed energy loss dE/dx of partons traversing the produced dense matter.

Main Objectives

The HIJING event generator is based on PYTHIA, ARIADNE and PDF libraries, and mainly used for event generation in experimental environment for baseline estimation. Since the todays programming techniques shifted to C++ based programming, the new generation of PYTHIA and PDF libraries are written in this language, furthermore, the experimental platforms are also shifting to C++ (e.g. AliRoot [4]), it is demanding to upgrade the HIJING accordingly. Hence, the main objectives are:

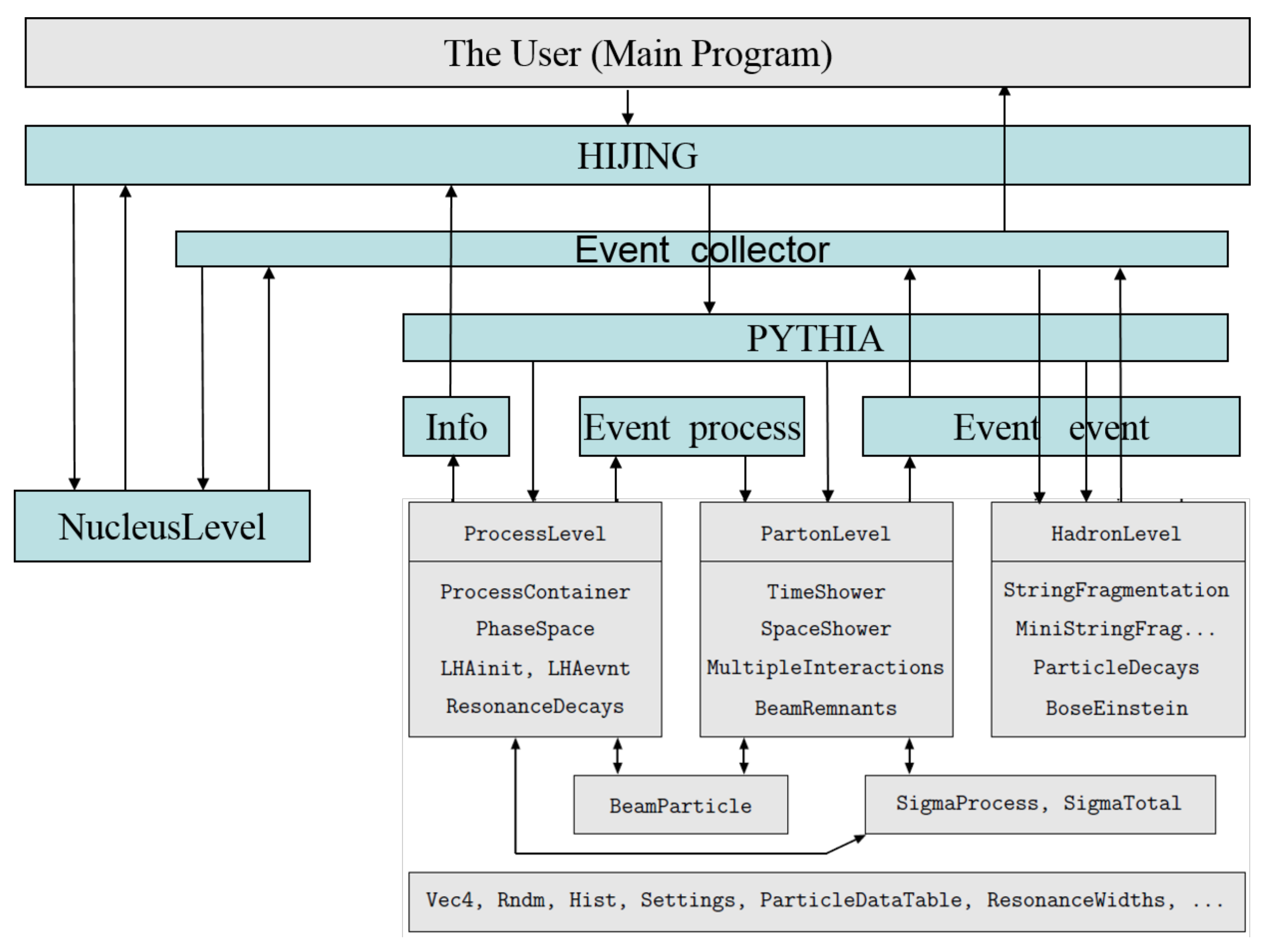
- write a genuine C++ based event generator,
- include the most recent public packages (like PYTHIA8 [3], LHAPDF6 [6]),
- support modularity:
  - possibility of inclusion/change to new theories, alternative processes (like DIPSY [10]),
  - possibility of alternative finite state processes (jet quenching models [9], etc.),
- introduce compatibility with experimental platforms (AliRoot),
- clean and upgrade of the HIJING model (e.g. Q^2 dependent shadowing [8, 11]),
- prepare the code for parallelization.

Since C++ is an object oriented language, it is more suitable for parallelization, being a longer term objective.

The program

The HIJING++ class is positioned in the PYTHIA8 namespace, using its functions to read configuration files (XML), PDF's. The PYTHIA random number generator may be replaced by other generators, with a possibility of GPU based random number generators. The main structure of the code is presented below:

```
namespace Pythia8 {

// HIJING class.
// Contains the top-level routines to generate an AA event.

class Hijing {

public:

   // Constructor. (See HIJING.cc file.)
   Hijing(string xmlDir = "../xmldoc",
          bool printBanner = true);

   // Destructor.
   ~Hijing();

   // Initialize.
   bool init();

   // Generate the next event.
   bool next();

   // The same read string function as in Pythia8
   bool readString(string, bool warn = true);

   // Auxiliary to set parton densities among list of possibilities.
   PDF* getPDFPtr(int idIn, int sequence = 1, string beam = "");

   ...

private:

   // The particle data table/database.
   ParticleData   particleData;

   // The event record for the nucleon information and the final state.
Event              event;
Event           final;
// Information on the generation: current subprocess and error
// statistics.
Info            info;
// Random number generator.
Rndm            rndm;
// Settings: databases of flags/modes/parms/words to control run.
Settings        settings;

// Class for handling the hard collisions
HardCollision  hijhard;

// Class for handling the soft pysics
SoftScatter    hijsoft;

// Class for handling the fragmentation
Fragmentation  fragmentation;

...
```



Structure of HIJING++.

Results

The first version of C++ code, based on the more recent HIJING version 2.552 [2] is ready and under test now. This version is based on PYTHIA8, while the soft physics is still the original ARIADNE version 4 based one. Hence, this version includes all the new physical processes included in the most recent version of PYTHIA. The original HIJING shadowing function was replaced by a HOPPET [8] based Q^2 evolved shadowing function. This modification softens the shadowing effect for larger Q^2 values.

The running time of the new version is comparable to the previous, FORTRAN based version, however, this version is suitable for parallelization, which may speed up the calculations considerably.

Forthcoming Development

As a next step, we try other random numbers generators, both CPU and GPU based ones. Also, the modularity will be extended, interfaces are to be defined at each step to the program, where new modules could be inserted, or change the original ones, thus allowing the partial change of models.

Since the code is written completely in C++ a (partially) GPU supported version of the program will be also tested. The preliminary investigations showed, that changing the random number generator to a GPU based version results only in a slight increase of the speed, depending on the GPU speed.

References

[1] X.N. Wang, M. Gyulassy, Phys. Rev. **D44**, 3501 (1991).
[2] W.T. Deng, X.N. Wang, R. Xu, Phys. Rev. **C83**, 014915 (2011).
[3] T. Sjöstrand, Comput. Phys. Commun. **191**, 159 (2015).
[4] http://aliweb.cern.ch/Offline/AliRoot/Manual.html
[5] L. Lönnblad, Comput. Phys. Comm. **71**, 15 (1992).
[6] http://lhapdf.hepforge.org/
[7] X.N. Wang, Phys. Rev.**C61**, 064910 (2001).
[8] A. Vogt, S. Moch, J.A.M. Vermaseren, Nucl. Phys. **B691**, 129 (2004)
[9] M.Gyulassy, P.Levai, I.Vitev, Phys .Rev. Lett. **85**, 5535 (2000).
[10] C. Flensburg, Prog.Theor.Phys.Suppl. **193**, 172 (2012).
[11] G. Ma, Sz. Harangozó, G. Papp, X-N. Wang, B-W. Zhang, in preparation

Acknowledgements

This work is supported by the Hungarian-Chinese cooperation grant No TéT 12 CN-1-2012-0016 and No. MOST 2014DFG02050, Hungarian National Research Fund (OTKA) grant NK106119. Author G.G. Barnaföldi also thanks the János Bolyai Research Scholarship of the Hungarian Academy of Sciences. We acknowledge the support of Wigner GPU laboratory, and discussions with Miklós Gyulassy and Wei-Tian Deng.