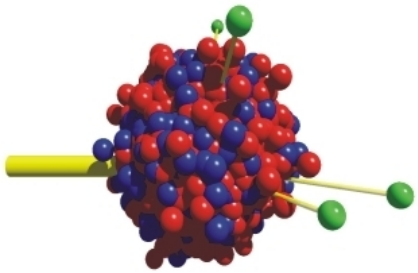
GiBUU: The Big Picture

Janus Weil

FIAS

Workshop on Transport Theory in Heavy Ion Collisions
Schmitten, 15.07.2013



GiBUU
The Giessen Boltzmann-Uehling-Uhlenbeck Project

Outline

GiBUU physics presented at this workshop:

- strangeness ($\rightarrow$ Laura)
- neutrinos etc ($\rightarrow$ Kai)
- dileptons ($\rightarrow$ Manuel)

this talk: 'big picture' instead of 'dirty details'

- biggest problems & challenges in hadronic transport
- possible solutions
- the course taken by GiBUU
- unique features of the model

questions:

- do we have a strategy for the future?
- how do we want to do transport in 5 years from now?

Problems & Challenges for the transport community

- **complexity**
  - large and old codes, O(100.000) LOC
  - most codes technically stuck in the 70s/80s
  - hard to maintain/extend
- **manpower**
  - mostly short-term contributions, few long-term maintainers
  - few young people
  - fragmented community
  - too little communication/collaboration
- too much **black-boxing**
  - no open development of codes
  - lacking reproducibility ("is it still science?")
  - lots of fighting (without conclusive results)
- often: missing **version control**
  - proper development & progress impossible!
- bad documentation

Answers / The Way Forward

- the transport community needs to arrive in the 21st century
- models need to be updated to **present-day programming languages** (C++, Python, modern Fortran)
- embracing modern computing concepts
  (object orientation, parallelization, etc)
- usage of **proper tools**:
  - version control (git, svn, ...) $\Rightarrow$ reproducibility, collaboration
  - documentation systems (RoboDoc, doxygen, ...)
  - bug trackers (bugzilla, trac, ...)
  - collaborative tools, wikis, ...
- we need **open codes** (and open development!)
- change in attitude (?)

Revolution vs Evolution

two possible ways to proceed:

1. "**Revolution**"
   - discard present models, start from scratch
   - many opportunities: pick new implementation language (e.g. C++); start with clean, well thought-out structure; get rid of historic ballast
   - downsides: major effort, will take lots of time; redoing work that has been done before; you will not do much (new) physics for some time
2. "**Evolution**"
   - why re-invent the wheel?
   - continuous improvement of present models
   - stick to Fortran, but move from old F77 to modern F95/2003/2008.
   - advantages: you don't start from zero; you can keep parts of the old code (well-tested & optimized); step-by-step upgrade; you can still do physics along the way
   - important: find 'sweet spot' between conservation and renewal

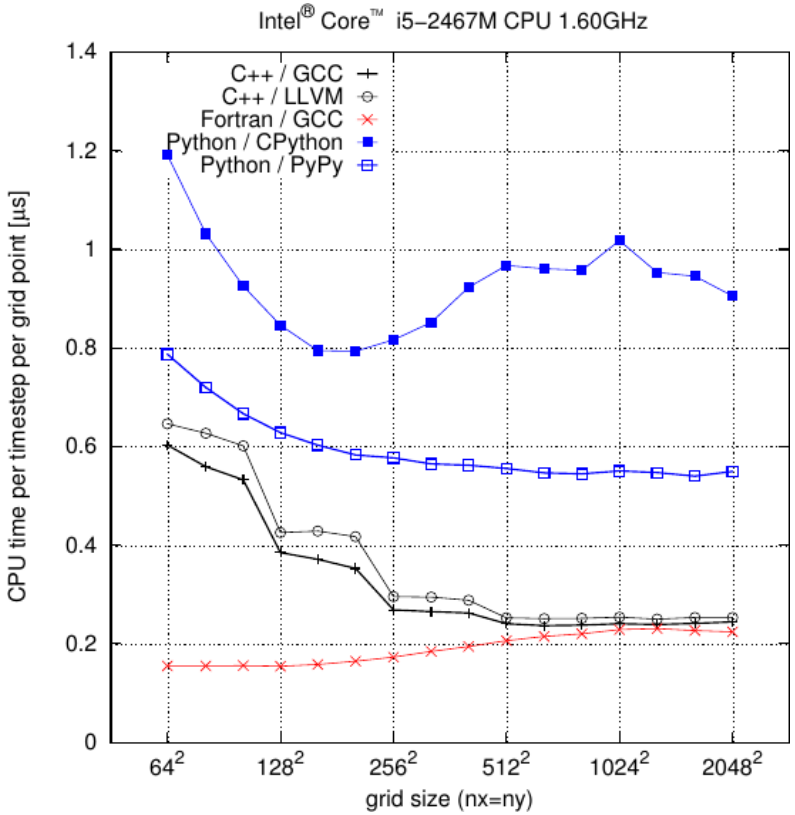
Fortran

- F77 (and earlier): "classical FORTRAN" ($\rightarrow$ UrQMD, HSD)
  - unstructured 'spaghetti' code, not very scalable
  - COMMON blocks, GOTOs, ...
- F90/95: modularization, derived types ($\rightarrow$ GiBUU)
- F2003: object orientation, polymorphism
- F2008: coarrays (parallelization)

- Fortran is a good language for numerics after all (vector notation, intr. complex type & exp. operator, ...)
- good performance, intr. parallelization support (F08)
- C++ is much more popular (multi-purpose language)
- modern Fortran is the "proper tool" for numerical simulations
- plus: it provides backward compatibility with 'legacy' code
- conclusion: Fortran is not a dead end! (sticking to F77 is)

Performance

- arXiv:1301.1334, S. Arabas et al., "Object-oriented implementations of the MPDATA advection equation solver in C++, Python and Fortran"
- same algorithm implemented in C++, Python, Fortran 2003 (using object orientation)
- result: Fortran shows best performance for all setups



GiBUU: basic facts

- result of $\sim$ 25 years of transport research in Giessen
- collaborative effort, large number of people have contributed over the years
- current contributors: K. Gallmeister, J. Weil, A. Larionov, T. Gaitanos
- manager & coordinator: Ulrich Mosel

- physics: purely hadronic (no partonic phase, no hydro)
- very versatile: unified framework for various reaction types (γA, eA, νA, pA, πA, AA)
- historically not as focused on AA as other models, more oriented towards elementary reactions ($\rightarrow$ Kai's talk)
- lot of work done in the low-energy regime
- review paper: O. Buss et al., Phys. Rept. 512 (2012)

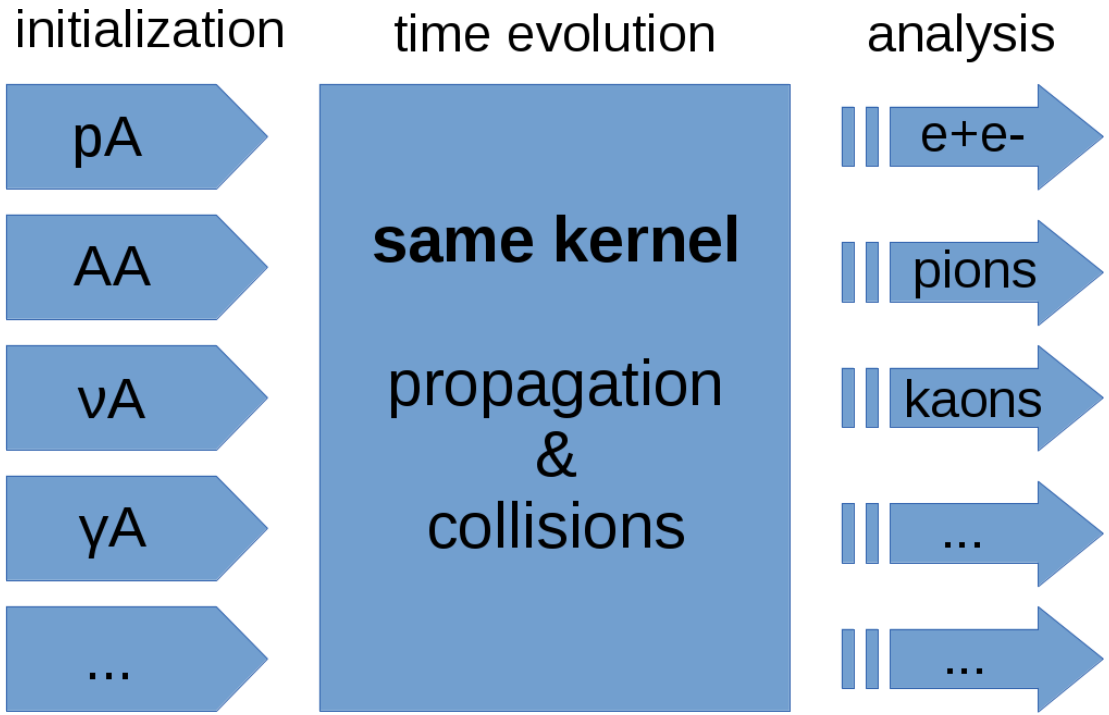
The BUU Equation

- BUU equ.: space-time evolution of phase-space density F (via gradient expansion from Kadanoff-Baym)

$$\frac{\partial(p_0 - H)}{\partial p_\mu}\frac{\partial F(x,p)}{\partial x^\mu} - \frac{\partial(p_0 - H)}{\partial x_\mu}\frac{\partial F(x,p)}{\partial p^\mu} = C(x,p)$$

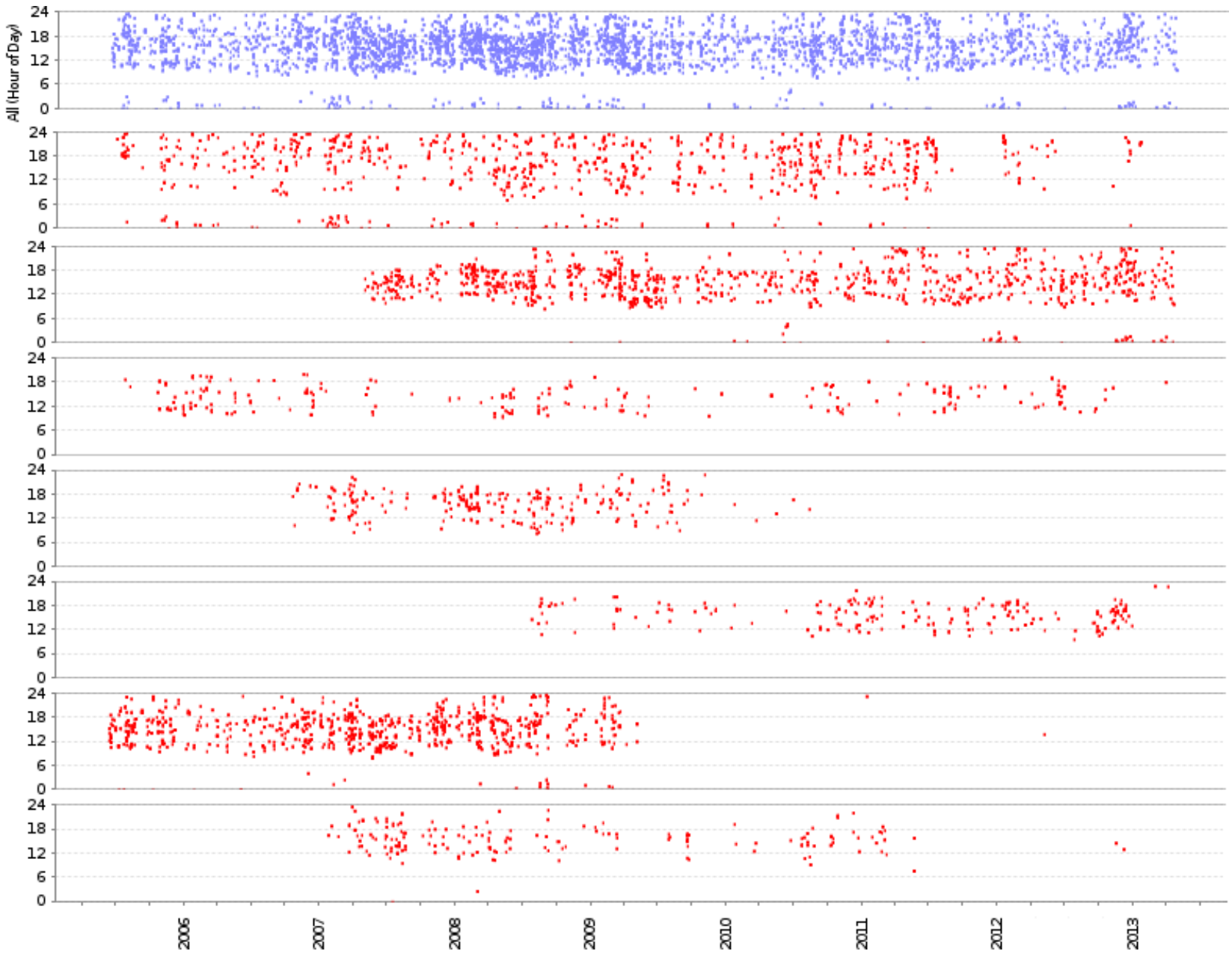
- Hamiltonian H:
  - hadronic mean fields (Skyrme, RMF)
  - Coulomb, "off-shell potential"
- collision term $C(x,p)$:
  - decays and scattering processes (2- and 3-body)
  - low energy: resonance model, high energy: string fragment.
- O. Buss et al., Phys. Rep. 512 (2012), http://gibuu.physik.uni-giessen.de

Model Overview

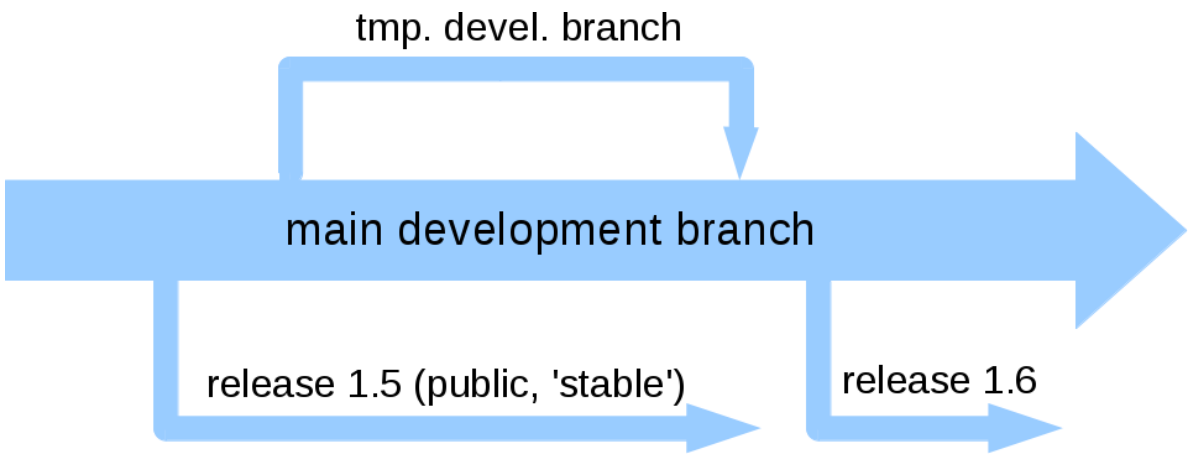


- basic idea: couple different initialization and analysis routines to same time-evolution kernel
- enough flexibility to cover all sorts of applications
- propagation options: cascade/Skyrme/RMF/off-shell
- collision term: Res. Model/Pythia/Fritiof
- any improvement in kernel will benefit all scenarios

GiBUU: commit activity (2005-2013)



Development Model / Releases



- basically all development happens on main branch
- several people work in parallel on same branch
- extra devel. branches only created for large/disruptive projects
- public releases:
  - 'stable', bugfixing only
  - some features removed
  - available upon request

Version Control

- important for:
  - reproducibility (e.g. specific version used in particular paper)
  - 'archeology' ("where did this line of code come from?")
  - collaboration (several people working on the same code)
  - managing a unified code version (preventing unnecessary fragmentation and extensive branching of the project)
- specific implementation not extremely important (git/svn/...)
- svn fits well with GiBUU strategy of having a single devel. branch (and not much branching)

Open Source

public availability of different codes:

	availability	date	version-controlled?
UrQMD	public	2010	no
GiBUU	"upon request"	2012	yes
HSD	"upon request"	2005	no

in principle every paper should come with:

- exact version info (model ABC, version x.y.z / revision 1234)
- input parameters used ('jobcard')

code should be available in order to:

- reproduce results if needed
- provide implementation details not given in the paper

Conclusions

1. GiBUU is a modern hadronic transport model (with quite a long history)
2. most well-maintained and versatile code available today
3. further work needed to clean up code base ($\rightarrow$ smooth upgrade path to F03/08)
4. aim: fully going open-source in the future ... (?)
5. contributors welcome!