HepMC and Rivet "challenges"

My perspective seen from the ALICE collaboration

HepMC and Rivet "challenges"

My perspective seen from the ALICE collaboration

Dual use of HepMC in ALICE

- Input for Rivet jobs
- Output of some event generators (EG)

Internally (old AliROOT and new O^2)

- For simulations, EG output converted to ROOT “standard” representation

Dual use of HepMC in ALICE

- Input for Rivet jobs
- Output of some event generators (EG)

Internally (old AliROOT and new O^2)

- For simulations, EG output converted to ROOT “standard” representation
  - Irrespective of EG output “format”

Dual use of HepMC in ALICE

- Input for Rivet jobs
- Output of some event generators (EG)

Internally (old AliROOT and new O^2)

- For simulations, EG output converted to ROOT “standard” representation
  - Irrespective of EG output “format”
    - HepMC output directly usable

Dual use of HepMC in ALICE

- Input for Rivet jobs
- Output of some event generators (EG)

Internally (old AliROOT and new O^2)

- For simulations, EG output converted to ROOT “standard” representation
  - Irrespective of EG output “format”
    - HepMC output directly usable
    - Other kinds of output (e.g., Fortran `COMMON` blocks) need custom conversion

Dual use of HepMC in ALICE

- Input for Rivet jobs
- Output of some event generators (EG)

Internally (old AliROOT and new O^2)

- For simulations, EG output converted to ROOT "standard" representation
  - Irrespective of EG output "format"
    - HepMC output directly usable
    - Other kinds of output (e.g., Fortran `COMMON` blocks) need custom conversion
- EG output (and all subsequent particles) written in ROOT representation

ROOT representation

Pros

- Highly efficient

  `TParticle` in `TClonesArray` branch of a `TTree`

Cons

ROOT representation

Pros

- Highly efficient

  `TParticle` in `TClonesArray` branch of a `TTree`
  - Binary format

Cons

ROOT representation

Pros

- Highly efficient

  `TParticle` in `TClonesArray` branch of a `TTree`
  - Binary format
  - Consecutive in memory

Cons

ROOT representation

Pros

- Highly efficient

  `TParticle` in `TClonesArray` branch of a `TTree`
  - Binary format
  - Consecutive in memory
- Scales

Cons

ROOT representation

Pros

- Highly efficient

  `TParticle` in `TClonesArray` branch of a `TTree`
  - Binary format
  - Consecutive in memory
- Scales
- Can *in principle* be fully parallelised

  E.g., calculate all p_{T}

Cons

ROOT representation

Pros

- Highly efficient

  `TParticle` in `TClonesArray` branch of a `TTree`
  - Binary format
  - Consecutive in memory
- Scales
- Can *in principle* be fully parallelised

  E.g., calculate all p_{T}

Cons

- Requires ROOT ($> 2\,\mathrm{GB}$ memory)

ROOT representation

Pros

- Highly efficient

  `TParticle` in `TClonesArray` branch of a `TTree`
  - Binary format
  - Consecutive in memory
- Scales
- Can *in principle* be fully parallelised

  E.g., calculate all p_{T}

Cons

- Requires ROOT ($> 2\,\mathrm{GB}$ memory)
- Based on `HEPEVT` `COMMON` block format

ROOT representation

Pros

- Highly efficient

  `TParticle` in `TClonesArray` branch of a `TTree`
  - Binary format
  - Consecutive in memory
- Scales
- Can *in principle* be fully parallelised

  E.g., calculate all p_{T}

Cons

- Requires ROOT ($> 2\,\mathrm{GB}$ memory)
- Based on `HEPEVT COMMON` block format
  - Conversion from HEPMC a (technical) challenge

ROOT representation

Pros

- Highly efficient

  `TParticle` in `TClonesArray` branch of a `TTree`
  - Binary format
  - Consecutive in memory
- Scales
- Can *in principle* be fully parallelised

  E.g., calculate all p_{T}

Cons

- Requires ROOT ($> 2\,\mathrm{GB}$ memory)
- Based on `HEPEVT COMMON` block format
  - Conversion from HEPMC a (technical) challenge
- No "standard" representation of $\langle meta \rangle$-data.

  b, Φ_R, ε, N_{part}, N_{coll}, ...

HepMC representation

Pros

- A standard

  in-so-far as it is followed

Cons

HepMC representation

Pros

- A standard

  in-so-far as it is followed

- Versatile

Cons

HepMC representation

Pros

- A standard

  in-so-far as it is followed

- Versatile
  - $\langle meta \rangle$-data

Cons

HepMC representation

Pros

- A standard

  in-so-far as it is followed

- Versatile
  - $\langle meta \rangle$-data
  - Allows “decoration” of particles and vertices

Cons

HepMC representation

Pros

- A standard

  in-so-far as it is followed

- Versatile
  - $\langle meta \rangle$-data
  - Allows “decoration” of particles and vertices
- Tree-structure

Cons

HepMC representation

Pros

- A standard

  in-so-far as it is followed

- Versatile
  - $\langle meta \rangle$-data
  - Allows “decoration” of particles and vertices
- Tree-structure
  - Allows for complicated queries

    Primary, prompt, from b, c, ...

Cons

HepMC representation

Pros

- A standard

  in-so-far as it is followed
- Versatile
  - $\langle meta \rangle$-data
  - Allows “decoration” of particles and vertices
- Tree-structure
  - Allows for complicated queries

    Primary, prompt, from b, c, …
- (Somewhat) modern implementation (HepMC3)

Cons

HepMC representation

Pros

- A standard

  in-so-far as it is followed
- Versatile
  - $\langle meta \rangle$-data
  - Allows “decoration” of particles and vertices
- Tree-structure
  - Allows for complicated queries

    Primary, prompt, from b, c, ...
- (Somewhat) modern implementation (HepMC3)
- Small memory footprint

Cons

HepMC representation

Pros

- A standard

  in-so-far as it is followed
- Versatile
  - $\langle meta \rangle$-data
  - Allows “decoration” of particles and vertices
- Tree-structure
  - Allows for complicated queries

    Primary, prompt, from b, c, ...
- (Somewhat) modern implementation (HepMC3)
- Small memory footprint

Cons

- Particle status codes not well standardised

HepMC representation

Pros

- A standard

  in-so-far as it is followed
- Versatile
  - $\langle meta \rangle$-data
  - Allows “decoration” of particles and vertices
- Tree-structure
  - Allows for complicated queries

    Primary, prompt, from b, c, ...
- (Somewhat) modern implementation (HepMC3)
- Small memory footprint

Cons

- Particle status codes not well standardised
- Beam representation ill-defined

HepMC representation

Pros

- A standard

  in-so-far as it is followed
- Versatile
  - $\langle meta \rangle$-data
  - Allows “decoration” of particles and vertices
- Tree-structure
  - Allows for complicated queries

    Primary, prompt, from b, c, $\ldots$
- (Somewhat) modern implementation (HepMC3)
- Small memory footprint

Cons

- Particle status codes not well standardised
- Beam representation ill-defined
- Tree-structure

HepMC representation

Pros

- A standard

  in-so-far as it is followed
- Versatile
  - $\langle meta \rangle$-data
  - Allows “decoration” of particles and vertices
- Tree-structure
  - Allows for complicated queries

    Primary, prompt, from b, c, ...
- (Somewhat) modern implementation (HepMC3)
- Small memory footprint

Cons

- Particle status codes not well standardised
- Beam representation ill-defined
- Tree-structure
  - Scales less well

HepMC representation

Pros

- A standard

  in-so-far as it is followed
- Versatile
  - $\langle meta \rangle$-data
  - Allows “decoration” of particles and vertices
- Tree-structure
  - Allows for complicated queries

    Primary, prompt, from b, c, ...
- (Somewhat) modern implementation (HepMC3)
- Small memory footprint

Cons

- Particle status codes not well standardised
- Beam representation ill-defined
- Tree-structure
  - Scales less well
  - Parallel computations harder

HepMC representation

Pros

- A standard

  in-so-far as it is followed
- Versatile
  - $\langle meta \rangle$-data
  - Allows “decoration” of particles and vertices
- Tree-structure
  - Allows for complicated queries

    Primary, prompt, from b, c, ...
- (Somewhat) modern implementation (HepMC3)
- Small memory footprint

Cons

- Particle status codes not well standardised
- Beam representation ill-defined
- Tree-structure
  - Scales less well
  - Parallel computations harder
- Non-particle states ill-defined

HepMC representation

Pros

- A standard

  in-so-far as it is followed
- Versatile
  - $\langle meta \rangle$-data
  - Allows “decoration” of particles and vertices
- Tree-structure
  - Allows for complicated queries

    Primary, prompt, from b, c, …
- (Somewhat) modern implementation (HepMC3)
- Small memory footprint

Cons

- Particle status codes not well standardised
- Beam representation ill-defined
- Tree-structure
  - Scales less well
  - Parallel computations harder
- Non-particle states ill-defined
- I/O extremely slow

HepMC representation

Pros

- A standard

  in-so-far as it is followed
- Versatile
  - $\langle meta \rangle$-data
  - Allows “decoration” of particles and vertices
- Tree-structure
  - Allows for complicated queries

    Primary, prompt, from b, c, …
- (Somewhat) modern implementation (HepMC3)
- Small memory footprint

Cons

- Particle status codes not well standardised
- Beam representation ill-defined
- Tree-structure
  - Scales less well
  - Parallel computations harder
- Non-particle states ill-defined
- I/O extremely slow
- I/O very voluminous

HepMC representation

Pros

- A standard

  in-so-far as it is followed
- Versatile
  - $\langle meta \rangle$-data
  - Allows “decoration” of particles and vertices
- Tree-structure
  - Allows for complicated queries

    Primary, prompt, from b, c, ...
- (Somewhat) modern implementation (HepMC3)
- Small memory footprint

Cons

- Particle status codes not well standardised
- Beam representation ill-defined
- Tree-structure
  - Scales less well
  - Parallel computations harder
- Non-particle states ill-defined
- I/O extremely slow
- I/O very voluminous
  - Possible resolution: binary format

HepMC representation

Pros

- A standard

  in-so-far as it is followed
- Versatile
  - $\langle meta \rangle$-data
  - Allows “decoration” of particles and vertices
- Tree-structure
  - Allows for complicated queries

    Primary, prompt, from b, c, …
- (Somewhat) modern implementation (HepMC3)
- Small memory footprint

Cons

- Particle status codes not well standardised
- Beam representation ill-defined
- Tree-structure
  - Scales less well
  - Parallel computations harder
- Non-particle states ill-defined
- I/O extremely slow
- I/O very voluminous
  - Possible resolution: binary format
- Not used by all EGs

Some status codes, but not adequate

Existing status codes

- 1 “Final state”

  Fuzzy - whatever the EG thinks this is
- 2 Decayed
- 4 Beam (initial) particle
- EG-dependent status codes

Ideally

- Finite set of status codes
- EG-specific $\langle meta \rangle$-info added as attributes

  PYTHIA adds “flow” attributes

Some status codes, but not adequate

Existing status codes

- 1 “Final state”

  Fuzzy - whatever the EG thinks this is
- 2 Decayed
- 4 Beam (initial) particle
- EG-dependent status codes

Preliminary Proposal

Ideally

- Finite set of status codes
- EG-specific $\langle meta \rangle$-info added as attributes

  PYTHIA adds “flow” attributes

Some status codes, but not adequate

Existing status codes

- 1 “Final state”

  Fuzzy - whatever the EG thinks this is
- 2 Decayed
- 4 Beam (initial) particle
- EG-dependent status codes

Ideally

- Finite set of status codes
- EG-specific $\langle meta \rangle$-info added as attributes

  PYTHIA adds “flow” attributes

Preliminary Proposal

- Define *finite* set of status codes
  - 1 Final state
  - 2 Decayed
  - 4 Initial
  - 5 Scattered
  - 6 Non-particle state
  - 7 Transport
  - ...

  EG people should decide

Some status codes, but not adequate

Existing status codes

- `1` "Final state"

  Fuzzy - whatever the EG thinks this is
- `2` Decayed
- `4` Beam (initial) particle
- EG-dependent status codes

Ideally

- Finite set of status codes
- EG-specific $\langle meta \rangle$-info added as attributes

  PYTHIA adds "flow" attributes

Preliminary Proposal

- Define *finite* set of status codes
  - `1` Final state
  - `2` Decayed
  - `4` Initial
  - `5` Scattered
  - `6` Non-particle state
  - `7` Transport
  - ...

  EG people should decide
- `HepMC3::GenParticle::set_status` *only* accepts these

Some status codes, but not adequate

Existing status codes

- 1 “Final state”

  Fuzzy - whatever the EG thinks this is
- 2 Decayed
- 4 Beam (initial) particle
- EG-dependent status codes

Ideally

- Finite set of status codes
- EG-specific $\langle meta \rangle$-info added as attributes

  PYTHIA adds “flow” attributes

Preliminary Proposal

- Define *finite* set of status codes
  - 1 Final state
  - 2 Decayed
  - 4 Initial
  - 5 Scattered
  - 6 Non-particle state
  - 7 Transport
  - ...

  EG people should decide
- `HepMC3::GenParticle::set_status` *only* accepts these
- *Everything else* use `HepMC3::GenParticle::add_attribute`.

Example: ALICE primary particles

A primary particle is a particle with a mean proper lifetime τ larger than $1\,\mathrm{cm/c}$, which is either a) produced directly in the interaction, or b) from decays of particles with τ smaller than 1 cm/c, restricted to decay chains leading to the interaction.

ALICE-PUBLIC-2017-005

Example: ALICE primary particles

> *A primary particle is a particle with a mean proper lifetime τ larger than $1\,\mathrm{cm/c}$, which is either a) produced directly in the interaction, or b) from decays of particles with τ smaller than 1 cm/c, restricted to decay chains leading to the interaction.*
>
> ALICE-PUBLIC-2017-005

In practise

- For every particle, go up tree until
  - parent with $\tau > 1\,\mathrm{cm/c} \rightarrow$ not primary
  - parent didn't decay (status 2) $\rightarrow$ not primary
  - parent is beam (status 4) $\rightarrow$ primary

Similar for *prompt* particles

HepMC paradigm

If particle $\tilde{p}$ changes, then it is a new particle

particles have zero or one start/end vertex

HepMC paradigm

If particle $\tilde{p}$ changes, then it is a new particle

particles have zero or one start/end vertex

- In $X^* \rightarrow X + Y$ the two $X*$ and X are not the same

HepMC paradigm

If particle $\tilde{p}$ changes, then it is a new particle

particles have zero or one start/end vertex

- In $X^* \rightarrow X + Y$ the two $X*$ and X are not the same
- In $X + Y \xrightarrow{\text{elastic}} X + Y$ none are the same (or $Q = 0$?)

Tree-structure has its limitations

- Scales poorly to Heavy-Ion
  Deep searches

Tree-structure has its limitations

- Scales poorly to Heavy-Ion
  Deep searches
- Array operations on particles difficult
  Calculate p_{T} of all particles

Tree-structure has its limitations

- Scales poorly to Heavy-Ion
  Deep searches
- Array operations on particles difficult
  Calculate p_{T} of all particles
- “Pb is 82 protons and 126 neutrons”

Tree-structure has its limitations

- Scales poorly to Heavy-Ion
  Deep searches
- Array operations on particles difficult
  Calculate p_{T} of all particles
- “Pb is 82 protons and 126 neutrons”
  - Yes, but RIVET requires *exactly* two beam particles

Tree-structure has its limitations

- Scales poorly to Heavy-Ion
  Deep searches
- Array operations on particles difficult
  Calculate p_{T} of all particles
- “Pb is 82 protons and 126 neutrons”
  - Yes, but RIVET requires *exactly* two beam particles
- Two participants interact - how to?

Tree-structure has its limitations

- Scales poorly to Heavy-Ion
  Deep searches
- Array operations on particles difficult
  Calculate p_{T} of all particles
- “Pb is 82 protons and 126 neutrons”
  - Yes, but RIVET requires *exactly* two beam particles
- Two participants interact - how to?
  - Split nucleus? nucleons? What status code?

Tree-structure has its limitations

- Scales poorly to Heavy-Ion
  Deep searches
- Array operations on particles difficult
  Calculate p_{T} of all particles
- "Pb is 82 protons and 126 neutrons"
  - Yes, but RIVET requires *exactly* two beam particles
- Two participants interact - how to?
  - Split nucleus? nucleons? What status code?
  - Perhaps

$$\left.\begin{array}{rcl} B_1 & \to & B_{1.1} + p_1 \\ B_2 & \to & B_{2.1} + p_2 \end{array}\right\} \to \quad p_1 + p_2 \to \dots$$

  $B_1, B_2, p_1,$ and p_2 w/status 4.
  $B_{1.1}, B_{2.1}$ perhaps other status ("Spectator?")

Example: IS+Hydro+Particlisation

1. Particles from initial state (IS — Glauber?)
2. Hydro-evolution to hyper-surface
3. “Particlisation” on hyper-surface and possible transport

- Steps 1 and 3 relatively clear

Example: IS+Hydro+Particlisation

1. Particles from initial state (IS — Glauber?)
2. Hydro-evolution to hyper-surface
3. “Particlisation” on hyper-surface and possible transport

- Steps 1 and 3 relatively clear
- But what to do to connect 1 to 2 and 2 to 3?

Example: IS+Hydro+Particlisation

1. Particles from initial state (IS — Glauber?)
2. Hydro-evolution to hyper-surface
3. “Particlisation” on hyper-surface and possible transport

- Steps 1 and 3 relatively clear
- But what to do to connect 1 to 2 and 2 to 3?

Suggestion

$$\underbrace{i_1, \dots}_{\text{IS, status } 4} \rightarrow \underbrace{H}_{\text{Hydro particle, status } X} \rightarrow \underbrace{h_1, \dots}_{\text{Particlisation}} \rightarrow \dots$$

Slow I/O

The problem

- Plain-text format → very large files
  `gzip` helps a bit

Slow I/O

The problem

- Plain-text format → very large files
  `gzip` helps a bit
- Very slow to encode and decode
  Scales poorly to Heavy-Ion

Slow I/O

The problem

- Plain-text format → very large files
  `gzip` helps a bit
- Very slow to encode and decode
  Scales poorly to Heavy-Ion

A solution?

- Dedicated binary format
  ensure same “endianness” for portability

Slow I/O

The problem

- Plain-text format → very large files
  `gzip` helps a bit
- Very slow to encode and decode
  Scales poorly to Heavy-Ion

A solution?

- Dedicated binary format
  ensure same “endianness” for portability

. . . but

. . . an even better solution

. . . an even better solution

Run EG and “client” in same process

Event kept in-memory

Needs

. . . an even better solution

Run EG and “client” in same process

Event kept in-memory

Needs

- EG as callable library

. . . an even better solution

Run EG and “client” in same process

Event kept in-memory

Needs

- EG as callable library
- EG output to HepMC format
  or provides converter

. . . an even better solution

Run EG and “client” in same process

Event kept in-memory

Needs

- EG as callable library
- EG output to HepMC format
  or provides converter

With RIVET as client
At least factor 10 speed-up

Tested EGs: AMPT, Angantyr, EPOS-LHC, SMASH

Summary

With my ALICE hat on . . .

- Would like to see wider adoption of HEPMC

Summary

With my ALICE hat on . . .

- Would like to see wider adoption of HEPMC
  - Significantly cut down confusion and code dealing with EG “particularities”

Summary

With my ALICE hat on . . .

- Would like to see wider adoption of HEPMC
  - Significantly cut down confusion and code dealing with EG “particularities”
  - Personally recently worked with CRMC and SMASH people on this

Summary

With my ALICE hat on . . .

- Would like to see wider adoption of HEPMC
  - Significantly cut down confusion and code dealing with EG “particularities”
  - Personally recently worked with CRMC and SMASH people on this
- Would like to see finite set of status codes

Summary

With my ALICE hat on . . .

- Would like to see wider adoption of HEPMC
  - Significantly cut down confusion and code dealing with EG “particularities”
  - Personally recently worked with CRMC and SMASH people on this
- Would like to see finite set of status codes
  - Significantly cut down confusion dealing with EG “particularities”

Summary

With my ALICE hat on . . .

- Would like to see wider adoption of HEPMC
  - Significantly cut down confusion and code dealing with EG “particularities”
  - Personally recently worked with CRMC and SMASH people on this
- Would like to see finite set of status codes
  - Significantly cut down confusion dealing with EG “particularities”
- Would like if EGs provided callable libraries

Summary

With my ALICE hat on . . .

- Would like to see wider adoption of HEPMC
  - Significantly cut down confusion and code dealing with EG “particularities”
  - Personally recently worked with CRMC and SMASH people on this
- Would like to see finite set of status codes
  - Significantly cut down confusion dealing with EG “particularities”
- Would like if EGs provided callable libraries
  - Tremendous speed-up gains over HEPMC I/O

Summary

With my ALICE hat on . . .

- Would like to see wider adoption of HEPMC
  - Significantly cut down confusion and code dealing with EG “particularities”
  - Personally recently worked with CRMC and SMASH people on this
- Would like to see finite set of status codes
  - Significantly cut down confusion dealing with EG “particularities”
- Would like if EGs provided callable libraries
  - Tremendous speed-up gains over HEPMC I/O
- HEPMC provides . . .

Summary

With my ALICE hat on . . .

- Would like to see wider adoption of HEPMC
  - Significantly cut down confusion and code dealing with EG “particularities”
  - Personally recently worked with CRMC and SMASH people on this
- Would like to see finite set of status codes
  - Significantly cut down confusion dealing with EG “particularities”
- Would like if EGs provided callable libraries
  - Tremendous speed-up gains over HEPMC I/O
- HEPMC provides . . .
  - (Relatively) Clear means of communication between EG and client

Summary

With my ALICE hat on ...

- Would like to see wider adoption of HEPMC
  - Significantly cut down confusion and code dealing with EG “particularities”
  - Personally recently worked with CRMC and SMASH people on this
- Would like to see finite set of status codes
  - Significantly cut down confusion dealing with EG “particularities”
- Would like if EGs provided callable libraries
  - Tremendous speed-up gains over HEPMC I/O
- HEPMC provides ...
  - (Relatively) Clear means of communication between EG and client
  - At no cost, a plethora of benchmark analyses via RIVET