Optimisation of the RPC branch in the HGeant output

A. Mangiarotti
LIP-Coimbra

July 17, 2013

The present documents describes the attempts undertaken to reduce the volume of the data saved to disk in the RPC branch of the current version of HGeant. The present version of the `HGeantRpc` object is implemented as follows.

```
Int_t    trackNumber;      // GEANT track number
Float_t  trackLength;      // track length at the RPC gap.                                  [mm]
Float_t  loctrackLength;   // local track length (inside the gap).                          [mm]
Float_t  eHit;             // energy deposited in the RPC gap.                              [MeV]
Float_t  xHit;             // x at the RPC gap, in module ref. system (EBOX).               [mm]
Float_t  yHit;             // y at the RPC gap, in module ref. system (EBOX).               [mm]
Float_t  zHit;             // z at the RPC gap, in module ref. system (EBOX).               [mm]
Float_t  tofHit;           // time of flight at the RPC gap.                                [ns]
Float_t  momHit;           // momentum at the RPC gap.                                      [MeV/c]
Float_t  thetaHit;         // polar angle at the RPC gap, in module ref. system (EBOX).     [deg]
Float_t  phiHit;           // azimuthal angle at the RPC gap, in module ref. system (EBOX). [deg]
Short_t  detectorID;       // detector ID (codified sector/column/cell/gap info)
```

The access methods of the present version of the `HGeantRpc` object are implemented as follows.

```
// Functions setVariable
inline void   setTrack(Int_t atrackNumber) {trackNumber = atrackNumber;}
inline void   setDetectorID(Short_t adetectorID) {detectorID = adetectorID;}
inline void   setAddress(Int_t sec, Int_t col, Int_t cel, Int_t gap);
       void   setHit(Float_t axHit, Float_t ayHit, Float_t azHit, Float_t atofHit,
                     Float_t amomHit, Float_t eHit);
       void   setIncidence(Float_t athetaHit, Float_t aphiHit);
       void   setTLength(Float_t tracklength, Float_t loctracklength);

// Functions getVariable
inline Int_t  getSector(void) const;
inline Int_t  getColumn(void) const;
inline Int_t  getCell(void)   const;
inline Int_t  getGap(void)    const;

inline Int_t  getTrack(void)      {return trackNumber;}
inline Int_t  getDetectorID(void) {return detectorID;}
       void   getHit(Float_t& axHit, Float_t& ayHit, Float_t& azHit,
                     Float_t& atofHit, Float_t& amomHit, Float_t& aeHit);
// Optimized for digitizer.
       void   getHitDigi(Float_t& axHit, Float_t& atofHit, Float_t& amomHit,
                         Float_t& aloctracklength);
//
       void   getIncidence(Float_t& athetaHit, Float_t& aphiHit);
       void   getTLength(Float_t& atracklength, Float_t& aloctracklength);
inline Int_t  getNLocationIndex(void)   {return 4;}
inline Int_t  getLocationIndex(Int_t i);
```

In the present version, for each **gap** where energy has been deposited during particle transport, one such object is saved. Two more such objects are also saved corresponding to the entrance and the exit of the EBOX volume by each particle depositing some energy in at least one gap or producing any secondary particle depositing some energy in at least one gap. The information is redundant and not well organised. In fact, for the gaps of the same detector cell, some quantities like the track length from the target, the polar and azimuthal angles of incidence and the z coordinate do not change appreciably. Moreover, gaps belonging to the same detector cell, and hence not accessible independently in terms of readout signals, are scattered into several objects located at different positions inside the HMatrixCategory. For these two reasons, it has been proposed to reorganise the HGeantRpc object to merge into a single object all the four gaps belonging to the same detector cell and purge unnecessary information. The new object structure must satisfy three requirements.

1. Allow to retrieve the average information for all the variables of the old version,
2. Allow to optionally store the relevant information for each gap independently,
3. Maintain backward compatibility.

The last condition is more severe than it might look. In fact, the HGeantRpc object is hold by the HMatrixCategory, which is a wrapper of a root TClonesArray. In turn, TClonesArray does not call the private streamer of HGeantRpc, but rather an internal optimised streamer. This automatic streamer to evolve between different versions of the same object must find variables with the same name, which will, by default, contain the same quantities when an older version of the object is read. It has been decided to keep unchanged the old object with the same variables and variable names and use it for a full **cell** (instead than for a single gap) by adding new variables to store the relevant information of the gaps. The proposed new object structure is the following.

```
Int_t    trackNumber;          // GEANT track number
Float_t  trackLength;          // track length at the RPC gap/cell.                                   [mm]
Float_t  loctrackLength;       // local track length (inside the gap/cell).                           [mm]
Float_t  eHit;                 // energy deposited in the RPC gap/cell.                               [MeV]
Float_t  xHit;                 // x at the RPC gap/cell, in module ref. system (EBOX).                [mm]
Float_t  yHit;                 // y at the RPC gap/cell, in module ref. system (EBOX).                [mm]
Float_t  zHit;                 // z at the RPC gap/cell, in module ref. system (EBOX).                [mm]
Float_t  tofHit;               // time of flight at the RPC gap/cell.                                 [ns]
Float_t  momHit;               // momentum at the RPC gap/cell.                                       [MeV/c]
Float_t  thetaHit;             // polar angle at the RPC gap/cell, in module ref. system (EBOX).      [deg]
Float_t  phiHit;               // azimuthal angle at the RPC gap/cell, in module ref. system (EBOX).  [deg]
Short_t  detectorID;           // detector ID (codified sector/column/cell/gap info)
// New elements after Jan 2013.
Float_t  loctrackLength1;      // local track length (inside the gap).                                [mm]
Float_t  eHit1;                // energy deposited in the RPC gap.                                    [MeV]
Float_t  xHit1;                // x at the RPC gap, in module ref. system (EBOX).                     [mm]
Float_t  yHit1;                // y at the RPC gap, in module ref. system (EBOX).                     [mm]
Float_t  momHit1;              // momentum at the RPC gap.                                            [MeV/c]
Float_t  loctrackLength2;      // local track length (inside the gap).                                [mm]
Float_t  eHit2;                // energy deposited in the RPC gap.                                    [MeV]
Float_t  xHit2;                // x at the RPC gap, in module ref. system (EBOX).                     [mm]
Float_t  yHit2;                // y at the RPC gap, in module ref. system (EBOX).                     [mm]
Float_t  momHit2;              // momentum at the RPC gap.                                            [MeV/c]
Float_t  loctrackLength3;      // local track length (inside the gap).                                [mm]
Float_t  eHit3;                // energy deposited in the RPC gap.                                    [MeV]
Float_t  xHit3;                // x at the RPC gap, in module ref. system (EBOX).                     [mm]
Float_t  yHit3;                // y at the RPC gap, in module ref. system (EBOX).                     [mm]
Float_t  momHit3;              // momentum at the RPC gap.                                            [MeV/c]
Short_t  HGeantRpc_version;    // HGeantRpc class version when reading from file.
```

This structure gives the backward compatibility and the flexibility of operating in two modes.

- Mode a): all the relevant information of each gap is preserved. In this case, the variables loctrackLength, eHit, xHit, yHit and momHit are used for the first gap. All other variables of the old structure still being filled with average values for the cell (when appropriate) giving a reduction in the size of the stored RPC branch.

- Mode b): only average values for the cell are stored. In this case, only the old variables are used and filled with average values (when appropriate), all the new variables being filled with zeros. The ROOT compression algorithm should allow to at least partially suppress variables that are always zero reducing the final size of the stored RPC branch.

One major drawback of the following reorganisation is represented by the hits in the EBOX volume for which the variables added in the new version are always zero and never contain any useful information. For a track crossing all the four gaps of a cell (which happens most of the times) the expected reduction is from $4 \times 11.5 = 46$ to 27 32-bits words (i.e. 41%). However, if the two additional hits in the EBOX volume are taken into account, there is actually an increase from $6 \times 11.5 = 69$ to $3 \times 27 = 81$ 32-bits words (i.e. 17%). One has to rely on the efficiency of the ROOT compression algorithm to suppress those zeros and bring an improvement in the size of the stored RPC branch. While the suppression is almost granted in Mode a), when the variables added in the new version are always zero for all HGeantRpc objects, the performance in Mode b), when the variables added in the new version are zero only for hits in the EBOX volume, is dubious.

Depending on the HGeantRpc class version, the digitiser can decide if the object represents a gap or a cell and take appropriate action. Moreover the option has been preserved of storing all relevant gap wise information, should the need to do so arise in the future.

This opportunity of changing the HGeantRpc object has also been taken to rationalise the access methods with the goal of providing a more modular and flexible interface to the information stored: in general only the quantities that are really necessary should be retrieved. This might led to probably small performance improvements in other parts of the code, like the digitiser, where the information is accessed frequently. Access methods referring to the “Gap” are meant to be used in Mode a), while methods referring to the “Cell Average” are meant to be used in Mode b). The old access methods have also been preserved to allow the old user code to work with the old version of HGeantRpc.

```
// Functions setVariable
inline void    setTrack(Int_t atrackNumber) {trackNumber = atrackNumber;}
inline void    setDetectorID(Short_t adetectorID) {detectorID = adetectorID;}
inline void    setAddress(Int_t sec, Int_t col, Int_t cel, Int_t gap);
inline void    setVersion(Int_t aHGeantRpc_version) {HGeantRpc_version = aHGeantRpc_version;}
       void    setIncidence(Float_t athetaHit, Float_t aphiHit);
//
// OLD set functions for backward compatibility.
//
       void    setHit(Float_t axHit, Float_t ayHit, Float_t azHit, Float_t atofHit,
                      Float_t amomHit, Float_t eHit);
       void    setTLength(Float_t atracklength, Float_t aloctracklength);
//
// NEW HIT set functions.
//
       void    setHit(Float_t axHit, Float_t ayHit, Float_t azHit, Float_t atofHit,
                      Float_t amomHit, Float_t eHit, Float_t aloctracklength);
//
       void    setGap(Int_t nGap, Float_t axHit, Float_t ayHit, Float_t amomHit,
                      Float_t eHit, Float_t aloctrackLength);
inline void    setTLengthHit(Float_t atrackLength) {trackLength = atrackLength;};
inline void    setZHit(Float_t azHit) {zHit = azHit;};
inline void    setTofHit(Float_t atofHit) {tofHit = atofHit;};


// Functions getVariable
inline Int_t   getSector(void) const;
inline Int_t   getColumn(void) const;
inline Int_t   getCell(void)   const;
```

```
inline Int_t   getGap(void)    const;

inline Int_t   getTrack(void)      {return trackNumber;};
inline Int_t   getDetectorID(void) {return detectorID;};
inline Int_t   getNLocationIndex(void) {return 4;};
inline Int_t   getLocationIndex(Int_t i);
inline Int_t   getVersion(void) {return HGeantRpc_version;};
//
//  OLD get functions for backward compatibility.
//
       void    getIncidence(Float_t& athetaHit, Float_t& aphiHit);
       void    getTLength(Float_t& atracklength, Float_t& aloctracklength);
       void    getHit(Float_t& axHit, Float_t& ayHit, Float_t& azHit,
                        Float_t& atofHit, Float_t& amomHit, Float_t& aeHit);
//
// NEW GAP get functions.
//
// Various options are available to avoid retrieving unnecessary information.
       Float_t getlocTLengthGap(Int_t nGap);
       void    getGap(Int_t nGap, Float_t& axHit, Float_t& ayHit, Float_t& amomHit,
                      Float_t& aeHit, Float_t& aloctrackLength);
       void    getGap(Int_t nGap, Float_t& axHit, Float_t& ayHit, Float_t& amomHit,
                      Float_t& aeHit);
       void    getGap(Int_t nGap, Float_t& axHit, Float_t& ayHit, Float_t& amomHit);
//
// NEW HIT get functions.
//
inline Float_t getTLengthHit(void) {return trackLength;};
inline Float_t getZHit(void) {return zHit;};
inline Float_t getTofHit(void) {return tofHit;};
// Various options are available to avoid retrieving unnecessary information.
       void    getHit(Float_t& axHit, Float_t& ayHit, Float_t& azHit, Float_t& atofHit,
                      Float_t& amomHit, Float_t& aeHit, Float_t& aloctrackLength);
       void    getHit(Float_t& axHit, Float_t& ayHit, Float_t& azHit, Float_t& atofHit,
                      Float_t& amomHit);
       void    getHit(Float_t& axHit, Float_t& ayHit, Float_t& azHit, Float_t& atofHit);
// Optimized for digitizer.
       void    getHitDigi(Float_t& axHit, Float_t& atofHit, Float_t& amomHit,
                          Float_t& aloctrackLength);
//
// NEW CELL get functions.
//
// Various options are available to avoid retrieving unnecessary information.
       void    getCellAverage(Float_t gap, Float_t& axHit, Float_t& ayHit, Float_t& azHit,
                              Float_t& atofHit, Float_t& amomHit, Float_t& aeHit,
                              Float_t& aloctrackLength);
       void    getCellAverage(Float_t& axHit, Float_t& ayHit, Float_t& azHit,
                              Float_t& atofHit, Float_t& amomHit, Float_t& aeHit);
       void    getCellAverage(Float_t& axHit, Float_t& ayHit, Float_t& azHit,
                              Float_t& atofHit, Float_t& amomHit);
       void    getCellAverage(Float_t& axHit, Float_t& ayHit, Float_t& azHit,
                              Float_t& atofHit);
// Optimized for digitizer.
       void    getCellAverageDigi(Float_t gap, Float_t& axHit, Float_t& atofHit,
                                  Float_t& amomHit, Float_t& aloctrackLength);
```

The new version of `HGeantRpc` has been implemented in Hydra and the digitiser reorganised and restructured to cope in the most transparent way possible with the three different scenarios for the input: i) old version of `HGeantRpc`, ii) new version of `HGeantRpc` used to store all relevant gap

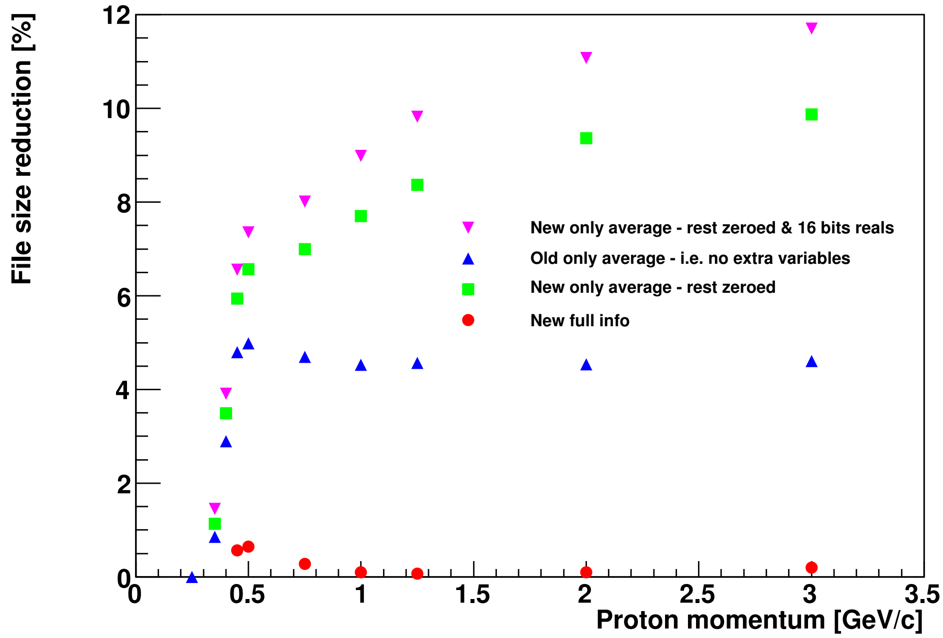

Figure 1: Reduction in percentage of the total file size for the four scenarios described in the text.

wise information and iii) new version of `HGeantRpc` used to store only average values for the cell. Backward compatibility has been tested extensively.

The final size of the generated files by HGeant has been tested using simplified events in which only one proton per sector with a well defined momentum was generated randomly selecting the azimuthal and polar angular ranges to uniformly illuminate the RPC wall. The results for the reduction in percentage of the total file size and in percentage of the RPC branch size are shown, as a function of the proton momentum, in Figure 1 and in Figure 2, respectively.

Four scenarios have been considered.

1. "New full info": This is Mode a) discussed previously.

2. "New only average - rest zeroed": This is Mode b) discussed previously.

3. "Old only average - i.e. no extra variables": This is a test with the old `HGeantRpc` object filled with average values for the full cell (when appropriate). In this case, the new variables are not even defined.

4. "New only average - rest zeroed & 16 bits reals": This is Mode b) discussed previously, but now all `Float_t` have been replaced with `Float16_t`, except that for the time of flight.

There is a slight tendency to reduce the RPC branch size, but the effect is not as big as one could have anticipated. The problem in scenario 1 is due to the hits in the `EBOX` volume that cause a wasting of space, as discussed. The `ROOT` compression algorithm works and in the end a small reduction is indeed observed. Scenario 2 brings a reduction, again because the `ROOT` compression algorithm works even better. The inversion of tendency with Scenario 3, which should be equal or better than Scenario 2, has no obvious explanation. It has to be kept in mind that in this case the structure of the output structure is changed and possibly the efficiency of the `ROOT` compression algorithm is different and strangely enough, lower. Finally, Scenario 4 is not as good as one could expect. A `Float16_t` is a special datatype of `ROOT` designed to be represented in memory with

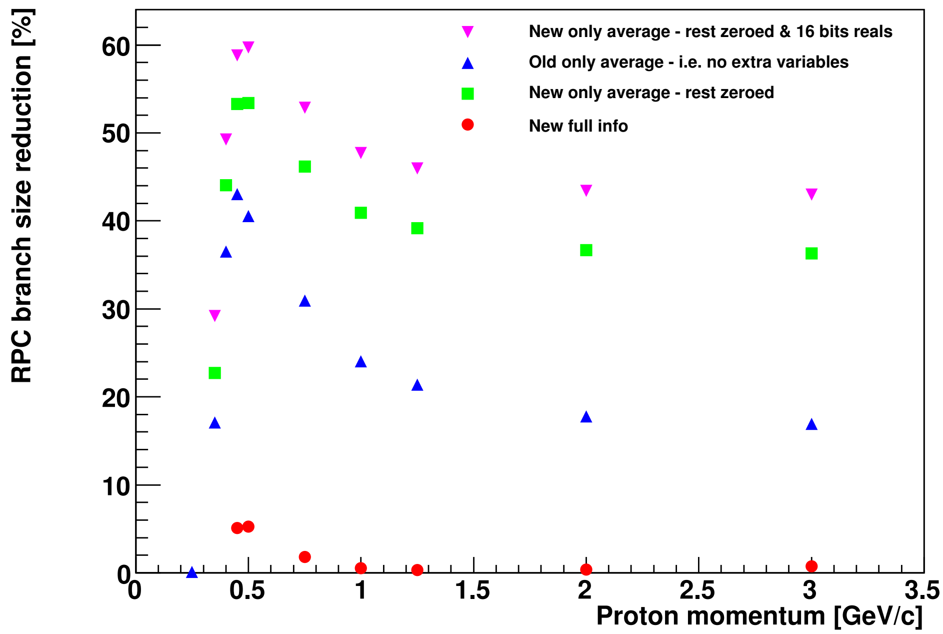

Figure 2: Reduction in percentage of the RPC branch size for the four scenarios described in the text.

scenario	reduction in total file size [%]	reduction in RPC branch size [%]
"New full info"	2	5
"New only average - rest zeroed"	15	35
"Old only average - i.e. no extra variables"	6	13
"New only average - rest zeroed & 16 bits reals"	18	42

Table 1: Reduction in total file size and in RPC branch size for an HGeant output obtained processing central Au+Au collisions at 1.23 AMeV simulated with UrQMD.

32-bits as `Float_t` but whose mantissa is truncated, when streamed to disk, so that a `Float16_t` occupies only 16-bits. It is to be expected that this compression at streaming would manifest fully only if the dedicated streamer of the `HGeantRpc` object is used, instead of the internal streamer of the `TClonesArray` class. Forcing the use of the dedicated streamer of the `HGeantRpc` object would require some modifications of the Hydra core and could bring to other performance penalties and has been avoided so far.

To fully asses the relevance of these results to a different situation typical of heavy ion collisions, events generated with UrQMD were also tested. They consisted of central Au+Au collisions at 1.23 AMeV. The reduction factors obtained under these conditions are reproduced in Table 1. The general trend is in agreement with the flat portion of the curves depicted in Figures 1 and 2. In detail, the scenarios 1 and 3 show the largest deviations. Again a different behaviour of the root streaming and compression algorithms is involved.