Buffered writing to output in the Clatterbridge simulation

Wednesday 7 February 2018

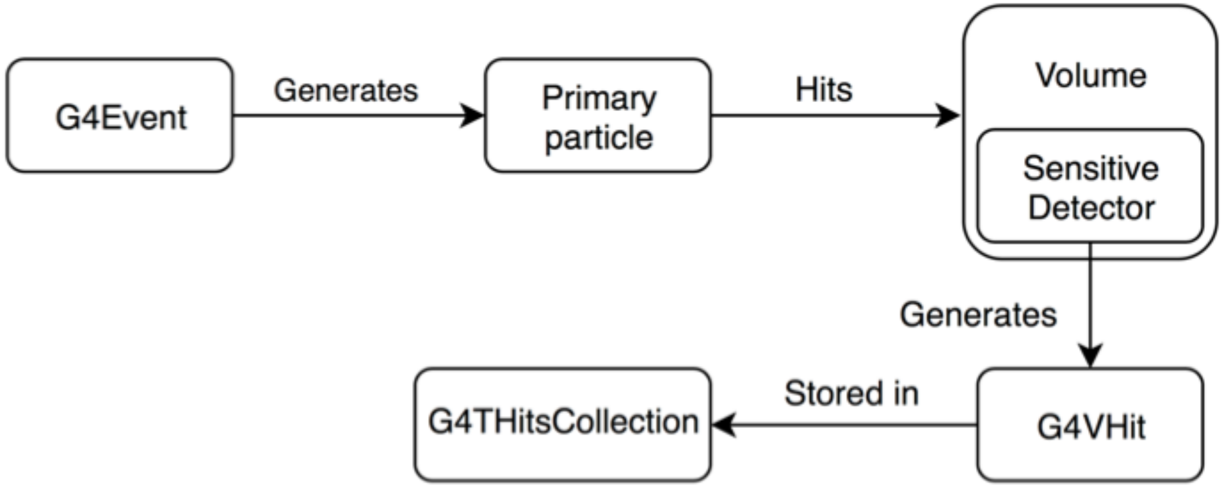
Matthieu Hentz

Why is buffered writing required?

- When a primary particle encounters a volume over which a detector (G4VSensitiveDetector) is defined, a **hit** (G4VHit) is produced
- The ProcessHits() method in the user implementation PhaseSpaceSD of the sensitive detector stores all hits occurring in an **event** (G4Event) in a **collection of hits** (G4THitsCollection)
- Dumping the hits at the end of every event is costly since the output file must be opened and closed each time (both slow operations)
- For a large number of events, time wasted becomes significant
- Dumping the hits in chunks should reduce time spent opening and closing files and speed up the simulation significantly

Issues with buffering (1/2)

- Process described on the previous slide



- A **run** consists of a given number of events, so is a set of repetitions of the above process
- **Pointers to hits** are stored in a hits collection. This leads to problems later.

Issues with buffering (2/2)

- To store hits that occurred within a run across events, a so-called **accumulator** is needed. It stores hits collections corresponding to different volumes in a std::vector.
- Need to populate the hits collections for the run at the end of each event: put hits on volume 0 in 0^{th} element of vector etc.
- Despite invoking a constructor every time a hit is saved in a hits collection in the SensitiveDetector, pointers to hits are assigned the same values over and over again (??) so many hits point to the same address in memory
- This leads to many of the hits in the final collections having the same values

Fixing issues by avoiding pointers

- Use dereference operator (*) and store values of hits in a vector of hits which itself is stored in a vector
- Hits can then be dumped from within the accumulator class and a parameter can be passed to choose the size of the buffer

Speedup

In macro: printModulo 100

# of events	No buffering	Buffer 10 events	Buffer 100 events	Buffer 1000 events
1,000	2.53s user 7.19s sys 9.751s total	2.45s user 3.42s sys 5.900s total	2.55s user 0.82s sys 3.413s total	2.39s user 0.41s sys 2.844s total
5,000	5.41s user 33.75s sys 39.207s total	4.73s user 14.59s sys 19.349s total	4.79s user 3.08s sys 7.976s total	4.67s user 1.07s sys 5.801s total
10,000	8.99s user 66.29s sys 1:15.32s total	7.55s user 29.68s sys 37.262s total	7.29s user 5.80s sys 13.164s total	7.24s user 1.82s sys 9.167s total

# of events	Buffer 10,000 events	Buffer 100,000 events
10,000	6.29s user 1.21s sys 7.535s total	
100,000	44.60s user 9.53s sys 54.180s total	48.71s user 12.05s sys 1:00.97s total