http://www.geant4.org

Bertini memory optimization

And bug fixing.

Andrea Dotti (adotti@slac.stanford.edu) ; SD/EPP/Computing,

Mike Kelsey

U.S. DEPARTMENT OF ENERGY
Office of Science

SLAC NATIONAL ACCELERATOR LABORATORY

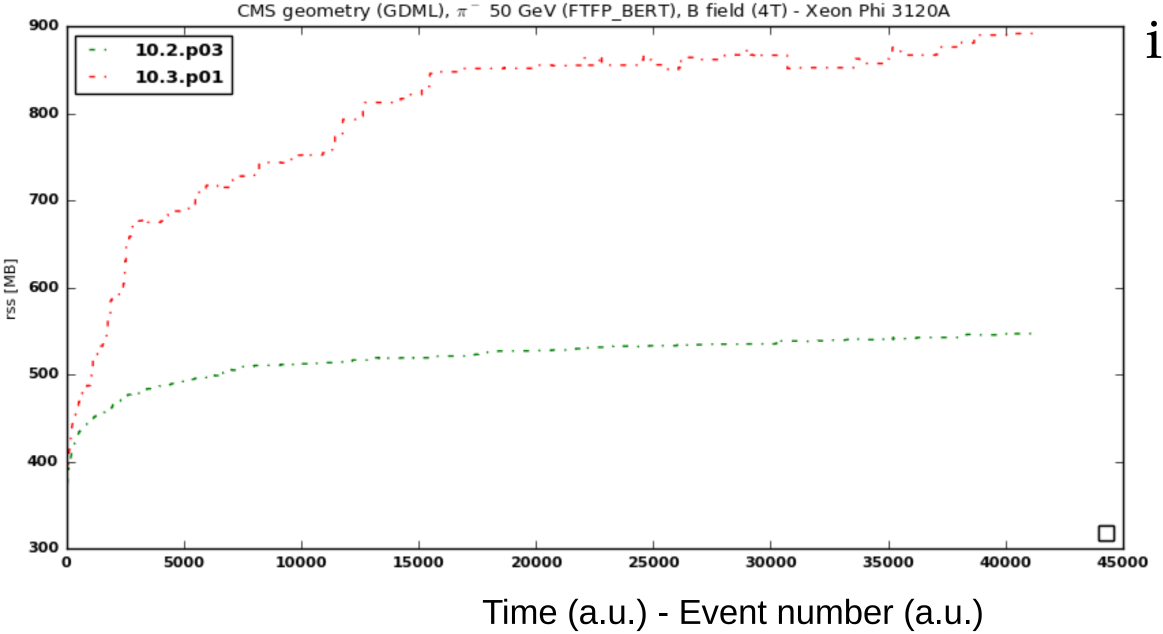
Outlook

Memory issue with 10.3

Analysis of issue

Recent fix

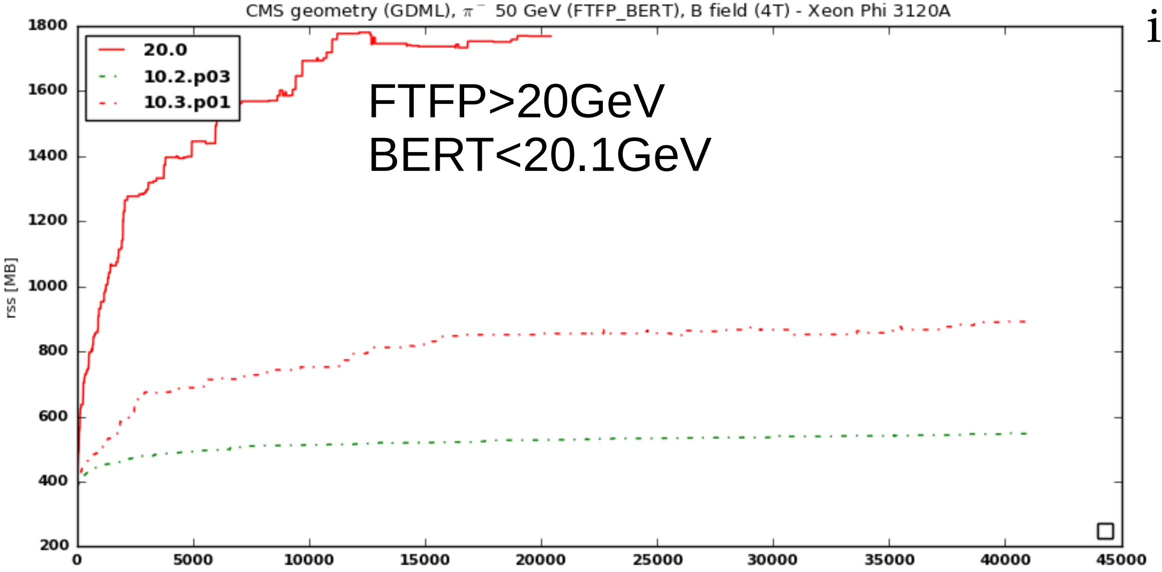
Memory profile in 10.3



Seen a substantial memory increase in 10.3:

- As a function of event number
- In our usual ParFullCMS tests
- **Linear with number of threads** (not shown here)

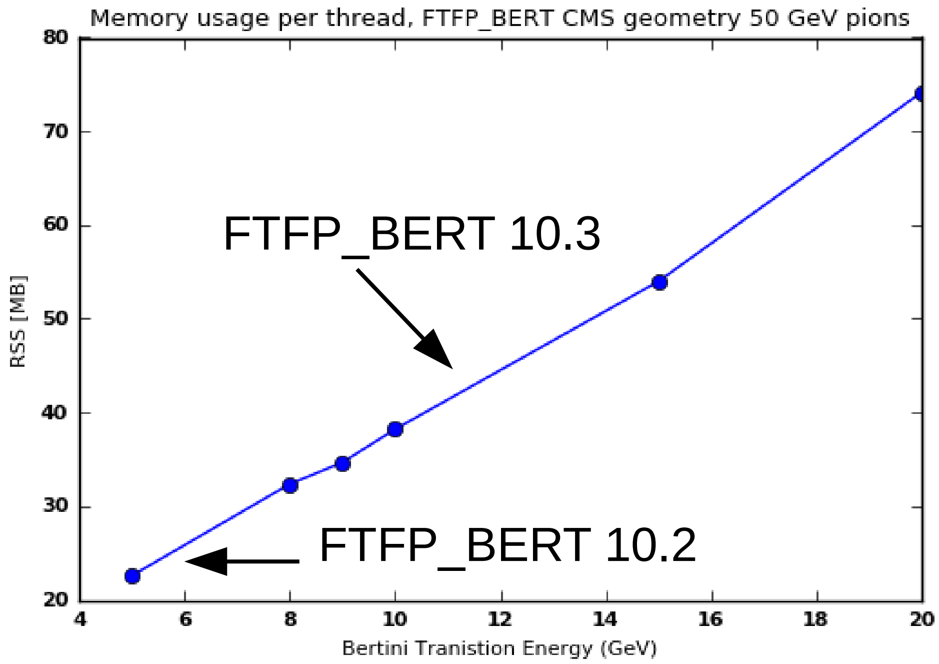
Memory profile in 10.3



Seen a substantial memory increase in 10.3:

- As a function of event number
- In our usual ParFullCMS tests
- Linear with number of threads
- **Linear with high-end of Bertini transition**

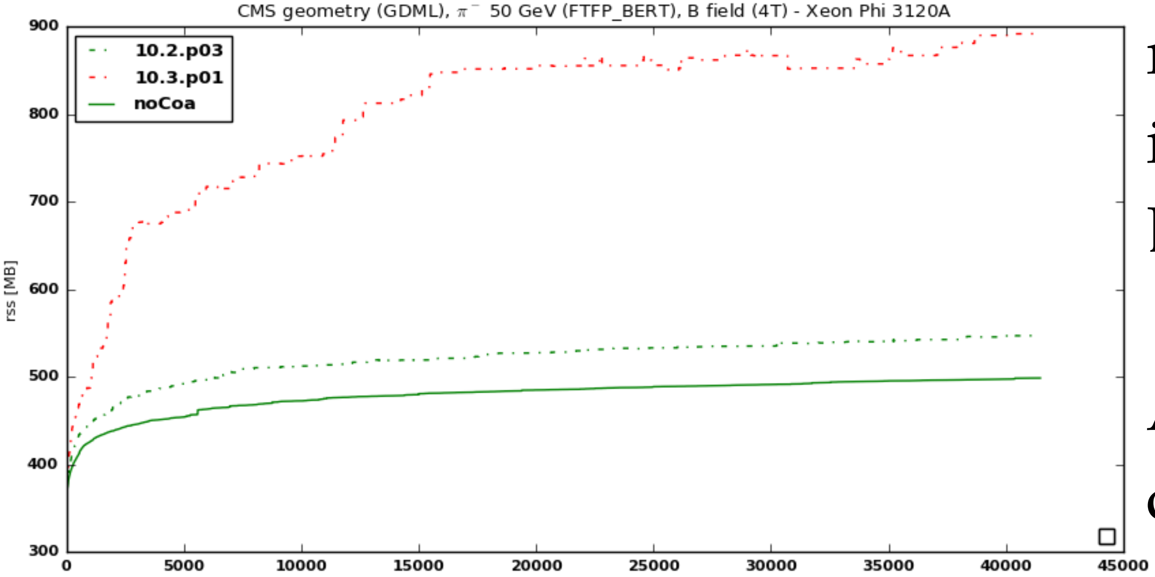
Memory profile in 10.3



Seen a substantial memory increase in 10.3:

- As a function of event number
- In our usual ParFullCMS tests
- Linear with number of threads
- **Linear with high-end of Bertini transition**

Memory profile in 10.3



After investigations realized the problem is in coalescence code in Bertini.

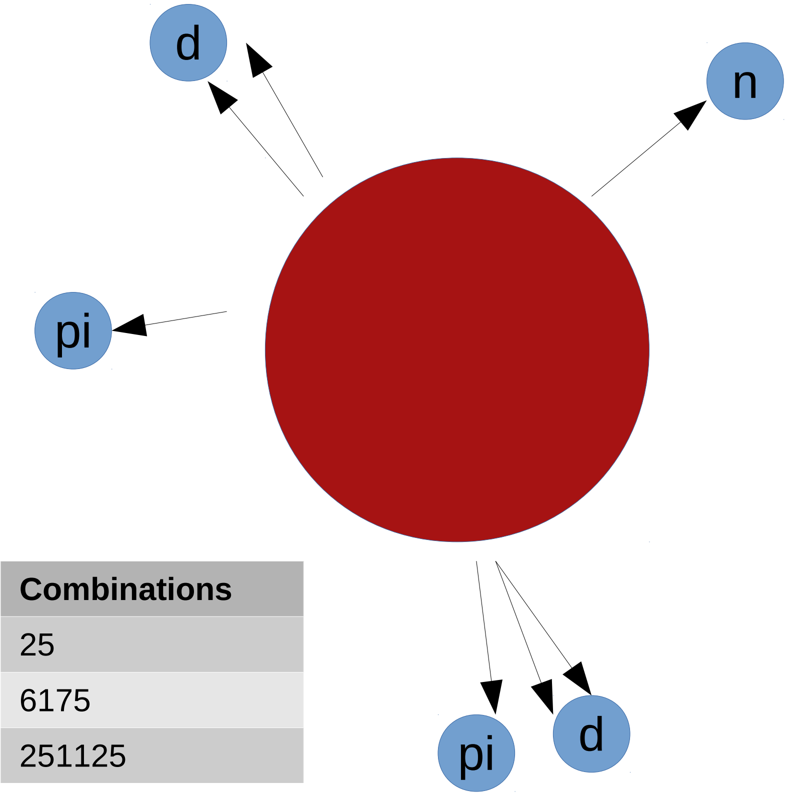
As extreme case: turn-off completely coalescence

Coalescence

After nuclear cascade: escaping nucleons are checked for possibility of forming a light fragment (d,t,He3 or alpha). The combinatorics of all exiting nucleons is considered:

$$\binom{n}{4} + \binom{n}{3} + \binom{n}{2}$$

# nucleons	Combinations
5	25
20	6175
50	251125



Coalescence

To avoid double counting a `std::set` was used to keep tracked of tried combinations

- an *hash* of the combination was used (using nucleon index) as input to the set
- only for very large multiplicities the size of the `std::set` can be actually observed
- in MT mode each thread has its own `std::set`, thus requiring memory proportional to the #threads

Bug in hashing function

During the debugging of the memory issue we also realized the hash function was bugged presenting high level of collisions

$$
\begin{aligned}
hash_4(n_0,n_1,n_2,n_3) = \quad & ((n_0 * 1000 + n_1) \\
& *1000 + n_2) \\
& *1000 + n_3 \\
hash_3(n_0,n_1,n_2) = \quad & \ldots \\
hash_2(n_0,n_1,n_2) = \quad & \ldots
\end{aligned}
$$

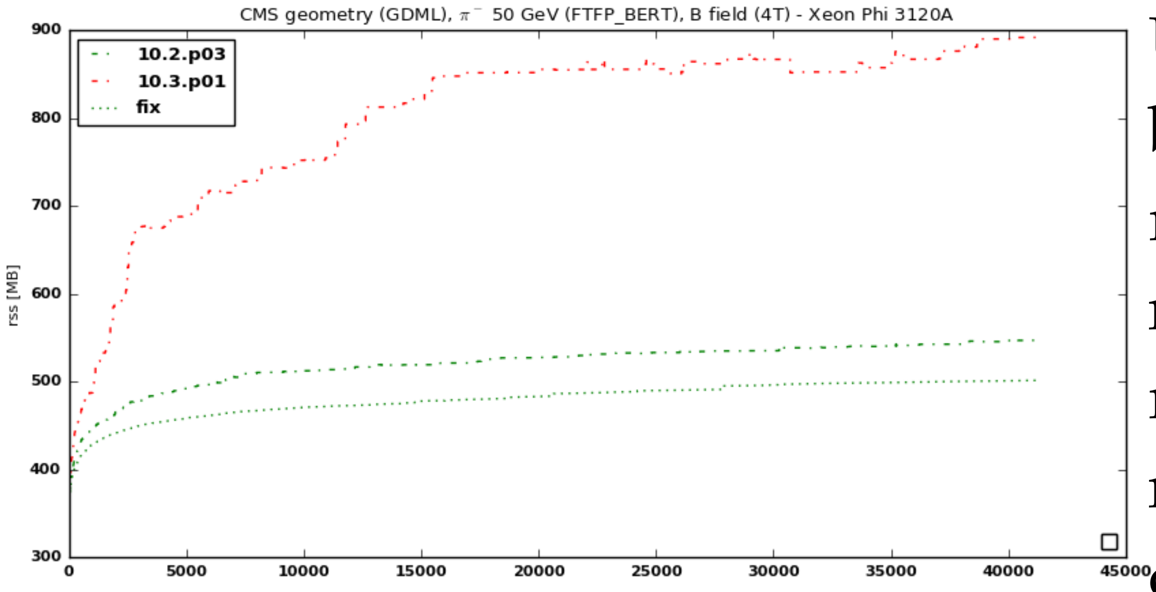
Hash$_4$(1,4,6,7) = 001004006007 = 1004006007

Hash$_4$(0,4,3,2) = 000004003002 = 4003002 ←

Hash$_3$(4,3,2) = 004003002 = 4003002 ←

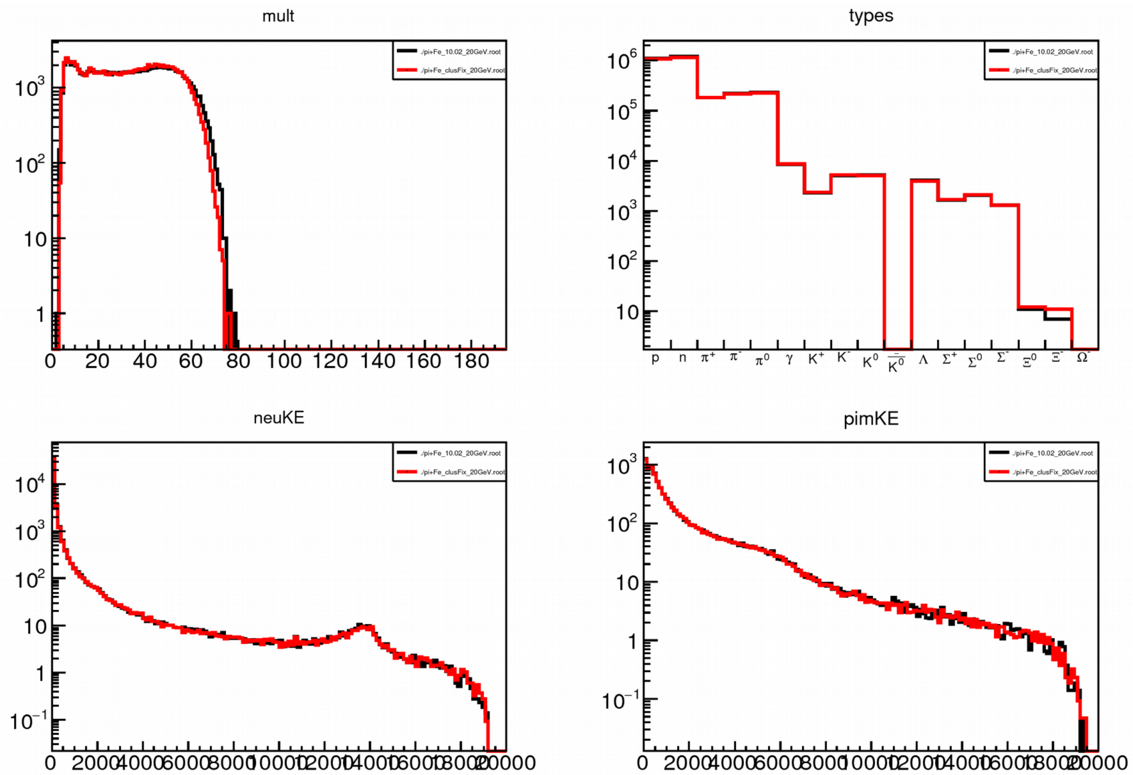
t/He3 candidate {4,3,2} is never considered

Fix and result



Use of `std::set` has been completely removed. Algorithm now is constant in memory and there is no need for the (bugged) cache

Fix and result



Expect minimal increase in number of light fragments produced and as consequence reduced multiplicity

Please note the very large multiplicities interactions: tens of nucleons to be considered was not so rare…

Why we did not see it with regular performance tests?

Memory is measured at the end of (selected) events in an event-action

At the beginning of each new interaction the `std::set` was cleared

- In sequential mode, there is only one `std::set`, we need to be (un-)lucky to have the last Bertini interaction to be a large multiplicity one to realize we have a problem
- In MT we measure memory from one thread, other threads can be in the middle of the simulation of a Bertini cascade, and thus we can have a "snapshot" of live memory

The sequential memory measurement is blind to abnormal use of memory in single event (we could measure memory in tracking action or some other trick like that), or simply use more MT

Conclusions

The memory increase seen in 10.3 was due to the coalescence code in Bertini

It was present also in older versions, but it appears only if:

- Bertini used at higher energies
- Multi-threading jobs
- High-statistics

Memory increase has been solved, together with a physics bug (artificial suppression of light fragments): **with this fix memory is even better than 10.2**

A high statistics memory test with large number of threads is needed to complete the tests performed by Soon: I will start to routinely perform this test on reference tags to be done on Xeon Phi with 100 threads