Fermilab

Managed by Fermi Research Alliance, LLC for the U.S. Department of Energy Office of Science

Towards modularization and vectorization of Geant4 hadronic physics: a pilot study

J.G.Lima (FNAL), S.Y. Jun (FNAL)
and T. Koi* (SLAC)

July 24 2018

* T.Koi is not active
in this project anymore

SLAC-FNAL pilot project on Geant R&D

Explore new computing avenues for hadronic physics simulation in HEP

Hadronic simulation is an important missing component of the GeantV vector prototype, which explores fine-grain parallelism using a top to bottom vectorization approach for particle transport simulation for next generation detector simulation.

Bertini cascade was chosen for this project, since it is the preferred model for low energy hadron-nucleus interactions, also it is relatively fast (compared to Binary cascade or INCL in Geant4) and it handles a large number of particle types.

- Provide standalone, vectorized Bertini algorithms (a specific hadronic cascade model)
- Modularized components (Geant4 and GeantV)
- Modern hardware technologies and parallel architectures

Project scope

- Modularize Geant4 Bertini cascade model and optimization – T.Koi (SLAC)
- SIMD vectorization of some computing-intensive algorithms – G. Lima (FNAL)
- Integration and computing performance evaluation – S.Y. Jun (FNAL)
- Identify requirements for future extension/development

Co-PIs: D. Elvira (FNAL) and A. Dotti (SLAC)

Implementation details and choices

- Use detailed profiling to identify CPU-heavy algorithms to demonstrate performance gains from vectorization
- Redesign data structures to promote vectorization with minimal overhead
- Use templated types to write generic algorithms to be instantiated using scalar or vector types as needed
- VecCore package to isolate the complexities of vectorization implementation from algorithms
- Benchmark every vectorized class, for close performance monitoring
- Validate physics simulation results with respect to Geant4

Profiler/OpenISpeedshop

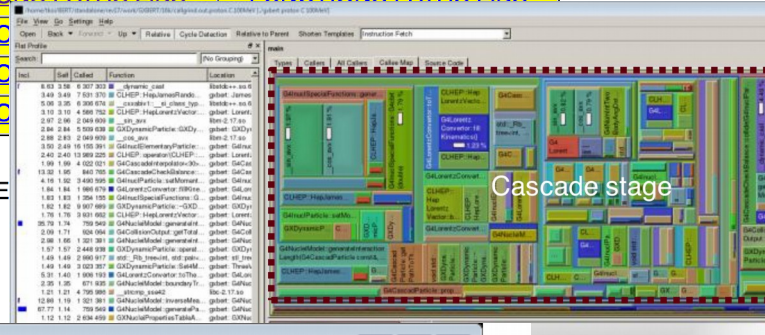
GXBERT (shared libs)

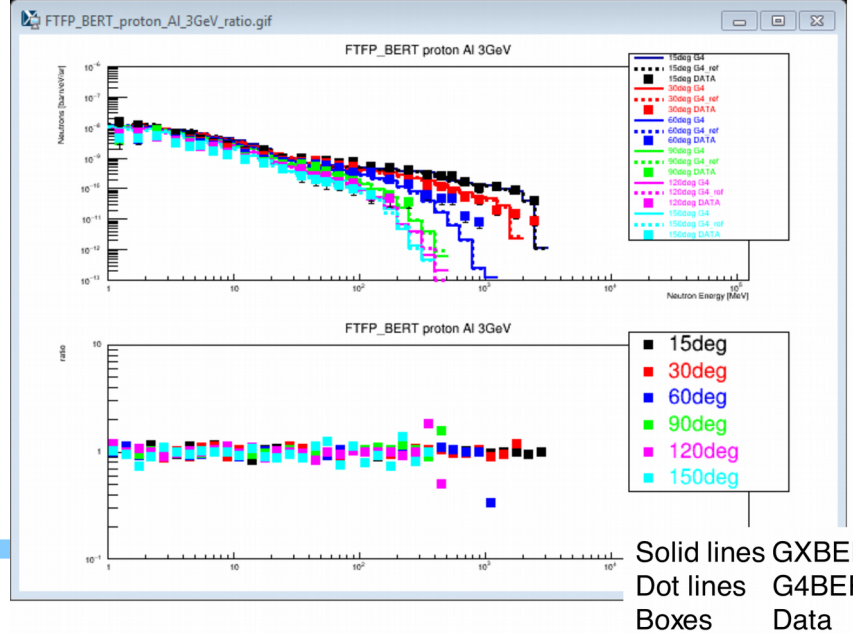
CPU profiling reports

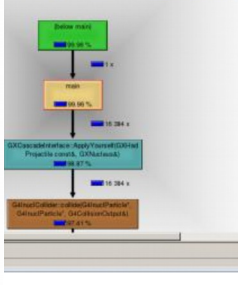
- FUNS: program counter @100Hz: EXCLUSIVE time for functions
- PATH: call path counter @35Hz: INCLUSIVE time for functions
- LIBS: libraries counter (LIBS)

Energy=1.5GeV	Carbon (C)	Lead (Pb)
gamma	FUNCTIONS PATHS LIBS	FUNCTIONS PATHS LIBS
pi-	FUNCTIONS PATHS LIBS	FUNCTIONS PATHS LIBS
proton	FUNCTIONS PATHS LIBS	FUNCTIONS PATHS LIBS
neutron	FUNCTIONS PATHS LIBS	FUNCTIONS PATHS LIBS
Lambda	FUNCTIONS PATHS LIBS	FUNCTIONS PATHS LIBS

Processor: Intel(R) Xeon(R) CPU E







Solid lines GXBERT
Dot lines G4BERT
Boxes Data

Two illustrative preliminary results

- Unit test for InuclElementaryParticle

```
lima@mac: build  ./TestInuclElementaryParticle
=== GXInuclElemParticles:  Particles=[proton; neutron; gamma; deuteron] masses=[938.272; 939.565; 0; 1875.61] types=<1 2 9 41> ekin=[1073.52, 925.289, 827.232
, 1155.82]
4 tracks: <1 2 9 41>
 kinE=[1073.52, 925.289, 827.232, 1155.82]
 totE=[1073.52, 925.289, 827.232, 1155.82]
 nucleon:m[1100]
 pion:m[0000]
 photon:m[0010]
 baryon:<1 1 0 2>
 strange:<0 0 0 0>
 quasi_deutron(): m[0000]
=== GXInuclElemParticles:  Particles=[pi+; pi-; diproton; dineutron] masses=[139.57; 139.57; 1876.54; 1879.13] types=<3 5 111 122> ekin=[1073.52, 925.289, 827
.232, 1155.82]
4 tracks: <3 5 111 122>
 kinE=[1073.52, 925.289, 827.232, 1155.82]
 totE=[1213.09, 1064.86, 2703.78, 3034.95]
 nucleon:m[0000]
 pion:m[1100]
 photon:m[0000]
 baryon:<0 0 2 2>
 strange:<0 0 0 0>
 quasi_deutron(): m[0011]
>>> GXInuclElementaryParticle tests passed.
```

- Benchmark for GXLorentzConvertor

```
lima@mac: build  ./LorentzConvertorBenchmark 3.0 1048576 10
GXBert results:   sumEscm = 1.96957e+09   sumEkin = 3.75451e+06   sumP2 = 9.23118e+11   CPUtime = 100.117
Scalar results:   sumEscm = 1.96957e+09   sumEkin = 3.75451e+06   sumP2 = 9.23118e+11   CPUtime = 63.2348
Vector size: 4
Vector results:   sumEscm = 1.96957e+09   sumEkin = 3.75451e+06   sumP2 = 9.23118e+11   CPUtime = 14.7479
VectorL result:   sumEscm = 1.96957e+09   sumEkin = 3.75451e+06   sumP2 = 9.23118e+11   CPUtime = 14.6649


GXBert results:   sumEscm = 1.96957e+09   sumEkin = 3.75451e+06   sumP2 = 9.23118e+11   CPUtime = 100.376
double results: sumEscm = 1.96957e+09   sumEkin = 3.75451e+06   sumP2 = 9.23118e+11   CPUtime = 63.1925
Double_v results: sumEscm = 1.96957e+09   sumEkin = 3.75451e+06   sumP2 = 9.23118e+11   CPUtime = 14.451
```

Some random thoughts

- Algorithm-level basketization seems like a good idea
  - e.g. assume uniform input arrays for Bertini cascade: [pp..p] on [Scint, Scint, Scint]
  - part of them will collide with Carbon atoms, others will collide with Hydrogen → rebasketize
  - at next level: rebasketize by physics process
  - at another level: based on N-body kinematics

    → algorithm-level basketization functionality
  - extra bonus: much better to satisfy requirements of event reproducibility

- Output arrays are not homogeneous
  - top level basketization will create homogeneous baskets (ok)
  - each particle should keep a “link” to unique “parent ID”, which could be its direct parent, or last persistent parent

- Challenging issues dealing with VcVector<int> and VcVector<double> in same algorithms
  - some new functionalities may be useful

Bertini vectorization status

- Git repository available: https://github.com/gxbert/gxbert.git
- Basic infrastructure for development, unit testing and performance evaluation (v01 done)
- New SoA data structure for tracks and kinematics (v01 done, extensions needed for nuclei)
- Vectorized ThreeVectors (a la CLHEP) and LorentzVectors (done)
- Basic algorithms for Lorentz boosts (Lab frame ↔ projectile ↔ center of mass frame) as needed (done)
  - measured speedup of ~3.4x in avx mode (theo.max = 4) w.r.t. scalar mode
  - additional 25% gain (scalar vs. G4), due to less branching and better memory locality
- Integration of our vectorized pRNG (pseudo-Random Number Generator) (done)
- Vectorized versions of base classes which define interfaces of hadronic Bertini models (under way for single-hadrons)
- Development and integration of multi-particle (vectorized) interfaces for algorithms (v01 done for single hadrons, extensions needed for nuclei)
- Currently vectorizing algorithm that handles interactions between elementary hadrons.
  - A long process, because of the large number of functions involved, some of them are rather difficult to vectorize due to branching and the triage process, x-section lookup, etc.
  - good chance of improvement if homogeneity of input vectors is assumed! (algorithm-level basketization?)
- Vectorization of particle-nuclei and nuclei-nuclei algorithms (not started yet)
- Assessment of performance gains and perspectives