GeantV: Parallelization for the future of particle transport simulations

Ryan Schmitz, EP-SFT, University of Minnesota

Advised by Sofia Vallecorsa, Andrei Gheata, (Mihaly Novak)
Plan of Attack

- What does our detector simulation software look like now? (Event-level parallelism, scalar processing)

- Where do we want to be? (Track-level parallelism, multi-particle processing (SIMD))

- What have I done to help us get there? (GeantV Calorimeter Application)
Background: What is Geant4?

- C++-based toolkit used to simulate the passage of particles through matter
- Geant4 is the dominant full detector simulation program in HEP and has been for years
- Initial release was in 1998, replacing Fortran-based Geant3
What’s the problem?

- Problem: Geant4 can't use processor cores efficiently (event-level vs. track level)
  - It can't be easily restructured to include this support

- Computers of the future will not have higher CPU speeds; they will have more cores

From GeantV Community Meeting, 10/2016
What’s the problem?

- Problem: Geant4 can't use processor cores efficiently (event-level vs. track level)
  - It can't be easily restructured to include this support

- Computers of the future will not have higher CPU speeds; they will have more cores

- Particle transport simulation is a natural application for parallel processing

From GeantV Community Meeting, 10/2016
Particle Transport is a Natural Fit for Parallelization

We should not wait for one event to finish…

CHEP 2012, New York, Re-thinking Particle Transport in the Many-Core Era
GeantV Structure
Geant V Speedup Potential

Current performance targets: **3x to 5x increase** (moving towards 10x or more in the long term)

Certain vectorized components already show an **order-of-magnitude** improvement over Geant4 (VecGeom)

Some full applications currently yield up to 5x speedup
- e.g. CMS application yields 3-5x speedup even when running in single-threaded mode
- More development must occur to achieve consistent speed gains

Above: CPU time required for three essential geometry navigation algorithms
Project status: Early June

A physics simulation toolkit must support three fundamental user-defined inputs:
Project status: Early June

A physics simulation toolkit must support three fundamental user-defined inputs:

- Incident Particles
  - Type
  - Energy
  - Direction

Particles (e-, e+, p, etc.)
Project status: Early June

A physics simulation toolkit must support three fundamental user-defined inputs:

- Incident Particles
  - Type
  - Energy
  - Direction

Physics configuration
- Physics list
- Production cuts
Project status: Early June

A physics simulation toolkit must support three fundamental user-defined inputs:

- Incident Particles
  - Type
  - Energy
  - Direction

- Physics configuration
  - Physics list
  - Production cuts

- Detector
  - Materials
  - Geometry

Particles (e-, e+, p, etc.)

Physics Config.

Detector (Calorimeter, Tracker, CMS/Atlas geometry, etc.)
Project status: Early June

A physics simulation toolkit must support three fundamental user-defined inputs:

- **Incident Particles**
  - Type
  - Energy
  - Direction

- **Physics configuration**
  - Physics list
  - Production cuts

- **Detector**
  - Materials
  - Geometry

Particles (e-, e+, p, etc.) → Physics Config. → Detector

Detector (Calorimeter, Tracker, CMS/Atlas geometry, etc.)

GeantV wasn’t defining this itself! (circa June 2017)
June Workflow: Defining and testing a calorimeter

Define calorimeter parameters
June Workflow: Defining and testing a calorimeter

- Define calorimeter parameters
- Create geometry in Geant4
June Workflow: Defining and testing a calorimeter

Define calorimeter parameters → Create geometry in Geant4 → Export geometry to ROOT
June Workflow: Defining and testing a calorimeter

1. Define calorimeter parameters
2. Create geometry in Geant4
3. Export geometry to ROOT
4. Load geometry from ROOT file
June Workflow: Defining and testing a calorimeter

1. Define calorimeter parameters
2. Create geometry in Geant4
3. Export geometry to ROOT
4. Load geometry from ROOT file
5. Run simulation, get results
June Workflow: Defining and testing a calorimeter

Define calorimeter parameters → Create geometry in Geant4 → Export geometry to ROOT

Load geometry from ROOT file
→ Run simulation, get results
→ Need to change geometry?
  Yes → Compile results for analysis
  No
June Workflow: Defining and testing a calorimeter

1. Define calorimeter parameters
2. Create geometry in Geant4
3. Export geometry to ROOT
4. Load geometry from ROOT file
5. Run simulation, get results
6. Compile results for analysis
7. Analyze data (output graphs, etc)

Yes
Need to change geometry?

No
June Workflow: Defining and testing a calorimeter

1. Define calorimeter parameters
2. Create geometry in Geant4
3. Export geometry to ROOT
4. Load geometry from ROOT file
5. Run simulation, get results
6. Need to change geometry?
   - Yes
     - Additional problems:
       - Running macros?
       - How to get/use geometry parameters?
   - No
     - Compile results for analysis
     - Analyze data (output graphs, etc)
Project Goals:

- Provide an example implementation of a user-defined detector, including geometry and materials

- Allow for this detector to be fully customizable without need to recompile

- Use “real physics” (as opposed to tabulated physics)
  - Tabulated physics: Results sampled from Geant4 sims
  - Real physics: Native GeantV models
GeantV Real Physics Calorimeter -- Project Result

- An application has been written which creates a user-defined calorimeter

- Application inputs:
  - Number of layers
  - Number of absorbers per layer
  - Absorber properties (thickness, material)
  - Particle beam inputs (type, energy)
  - Production cuts (length, energy)

- Application Outputs (per primary per absorber):
  - Energy deposited in each material
  - Charged/neutral track length in each material
  - Number of secondaries of each type produced (gammas/electrons/positrons)
    - *All of these outputs include std. deviation!*
Output comparison to Geant4 -- Perfect match
June Workflow: Defining and testing a calorimeter

1. Define calorimeter parameters
2. Create geometry in Geant4
3. Export geometry to ROOT
4. Load geometry from ROOT file
5. Run simulation, get results
6. Need to change geometry?
   - Yes
     - Additional problems:
       - Running macros?
         - Yes
           - How to get/use geometry parameters?
         - No
           - Compile results for analysis
   - No
     - Analyze data (output graphs, etc.)
Current Workflow: Defining and testing a calorimeter

Create geometry in GeantV → Define calorimeter parameters

Run simulation, get results → Analyze data (output graphs, etc)

Additional problems:

Running macros? *easy*

How to get/use geometry parameters? *Use get methods*
Summary

- Geant4 is important to computational physics, but can’t utilize the full potential of modern hardware

- GeantV is a promising improvement for the future, and it shows great potential for improved simulation speeds

- My contribution: A customizable calorimeter, fully native to GeantV, that serves as one of the first completely standalone examples of a user-defined GeantV application