Design, implementation and performance results of the GeantV prototype.

Andrei Gheata for the GeantV R&D team

Plan of the talk v0

Introduction, context, motivation

~8 slides

Introduction

- Present the R&D scope and goals
- Ideas
  - Grouping tracks doing same work
  - make the work vectorizable on tracks
  - Gather tracks from more events to increase populations
- Initial goals
  - Factor 2-5 coming mainly from increased locality and vectorization
  - Opening new opportunities for accelerators

Technology context (2 slides)

- Hardware evolution trend
  - Hitting the walls: power, ILP, memory access -> multi/many cores with SIMD pipelines
- Why track-level parallelism and vectorization
  - Making use of a larger % of modern machines in simulation

HEP context

- Run3->HL computing requirements
- Simulation still a bottleneck in many HEP pipelines
- Demand for simulated samples increasing with luminosity

Simulation context (2 slides)

- State of the art simulation application morphology
  - Low granularity OO (deep stacks, virtual calls), functionality & physics performance oriented design vs. performance
- Complexity of stepping
- Simulation pie chart (fraction of time in different stages)
- Access to code is a real issue (CPU is frontend bound)

Motivation (3 slides)

- Locality and vectorization – why?
- Motivations for taking a backward incompatible development path
  - Change of the data layout and flow, track-level parallelism
  - Preserve generality
- Accelerators – explore a model to port simulation to GPUs

Design considerations, concepts and architecture

(~10 slides)

Data processing oriented design

- Concept of baskets and functional locality
- Track as single placeholder for state
- Evolution of the basket concept
  - Common SOA shared by multiple threads
  - Thread-specific baskets and SIMD-aware baskets

Code redesign: vector interfaces

- Templated kernels reusable for scalar/vector data types
- Geometry locality and basket-aware navigation - VecGeom
- Enforcing short vectorization – evolution towards VecCore

First prototypes of scheduling (1 slide)

- Bottlenecks, lessons learned
  - SOA is not the universal cure in non-pipeline data flows
  - Very hard to do efficient concurrency and preserve locality at same time
  - Bundling too much work explodes memory
  - …

Going complex: algorithm-oriented design

- Scheduler v3, simulation stages
- Support for scalar and vector flow in a single framework
- Stacking is memory efficient, how to reuse it in a vector flow?

Concurrency design

- Minimize context switches: single kernel thread model (worker) vs. TBB-base attempts
- Event slots: a necessity…?
  - More data for same work
- Fine versus coarser grain parallelism: the optimum is in between
- External-driven parallelism
- Towards functional programming style: percolating all state (task data) through interfaces, making functions fully re-entrant

Accelerators: integration with the “basket” model

- Portability: CUDA as a backend
- GPU baskets and performance
- Lessons:
  - Portability is feasible, but does not come for free
  - efficiency comes with very large baskets, which are difficult to maintain
- Soon/Philippe: can you add some performance measurement, remarks/conclusions from our preliminary GPU studies?

- John commented: move to separate section

Implementation (10 slides)

Components supporting a general vectorization approach

VecCore – a portable vectorization library

- Solving portability and vectorization efficiency
- Layout and API, supported architectures
- Guilherme: can you add a summary slide for VecCore? Can be more in the backup, but at least one is important

VecGeom – vectorized geometry library

- Support for vectorization on multiple levels
- Multi-architecture
- Performance-driven development: best algorithms inspired from GeantV/ROOT/USolids
- Production-level quality
- Unit-test performance
- VecGeom team, can you provide a summary slide for the main features and a few backup slides with performance measurements, plans for evolving and integrating the package?

VecMath: vectorization support for math utilities

- RNG, fast math functions (vectorized)

Vectorized magnetic field (1-2 slides)

- Using templates rather than virtual inheritance
- Lane-refill driven integration policy for vectorization
- Some results
- Some lessons learned
- John, can you provide 1-2 slides with main messages, more for the backup if you want?

Vectorized EM physics models (2-3 slides)

- General features: multiple physics list support, compact implementations, vectorized sampling and rotation methods, …
  - Mihaly, do you want to make a separate slide before this one with general features of the physics implementation?
- Unit tests performance: vectorization can be efficient per model
- Lessons learned:
  - compactness brings much more benefits than vectorization for “small” hotspots
- Marilena, can you provide 1-2 slides with main physics vectorization features and lessons? We can have much more in the backup also.

MC truth: keeping track of kinematics (1 slide)

- Implemented as user hook based on filters
  - Start gathering MC info triggered/stopped by user
  - Concurrent merging service provided
  - Simple examples available
  - Could certainly be refined more, providing support for more complex use cases

- Witek, can you make one slide a out this please? Something more detailed for the backup won't hurt as well

User interfaces: a compromise (few slides)

- From:
  - give me your “hit” model, I give you factories and tools to handle and store them efficiently concurrently
    - Nice concurrent merging service ending up in ROOT (TBufferMerger)
- To:
  - Here are the hooks allowing to allocate your own data and providing per-thread handles
  - Here are some example data models making use of event slots
  - Here is the workflow allowing to score concurrently and merge hit information
  - Storing the hits or passing them to a digitization module is the user business

Integration with experiment frameworks

- Kevin, Sunanda, as we discussed, can you provide few slides about the integration effort and the main results? We can have many more details in the backup slides.

Performance results

Description of benchmarks, performance comparison plots, discussion (10 slides)

Benchmarks description (1-2 slides)

- TestEm3/5, FullCMS/LHCb
- Mention that we focused on CMS simulation as a more realistic LHC use case
- Describe different test modes: single track mode, basketization ON/OFF, …

Benchmarks physics results comparison

- Show that results are matching at per mil level
- Mihaly, can you make one slide with a plot and some comparison numbers?

Performance summary table

- Summary of speed-ups for different architectures
- Alberto, can you make a summary table with performance on different architectures? You can centralize it to Soon first since he will probably have complementary information.

Performance insight (3-4 slides)

- Profiles from VTune (microarchitecture analysis)
- Counters from Soon
- Discussion of benefits/overheads
- Soon + Philippe, we will need few slides with tables/plots, comparisons and their interpretation (facts but also speculations, since this is the state of our current understanding)

Concurrency performance (few slides)

- CPU scalability plot (Kevin, a scalability plot in CMSSW would be useful here)
- Memory scalability plot
- Discussion fine grain versus coarser grain
  - Does the assumption that you need to gather tracks from more events to have reasonable populations in baskets still stands?
  - Concurrent events versus concurrent baskets/tracks
  - Keeping tracks in producer threads is good, minimize stealing
  - Track memory fragmentation is a huge issue (not solved yet)

Everyone: I really count on your input here, from your own perspective

Lessons learnt (5 slides+)

Major lessons, what to do and not to do dealing with locality, concurrency, vectorization

Lessons and open questions

- Is basketization good, what are the limitations?
- How to deal with memory, track allocation policies.
- Is splitting the code in compact functional blocks good? Yes.
- What is the best concurrency granularity? Is track-level parallelism good?
  - Fine granularity concurrency introduces overheads. Avoid if not compelled to do so
  - Event/sub-event + functional regrouping of tracks is probably the best. Needs flexibility for processing ordering within a step or per category (e.g. tracks doing specific physics in specific geometry regions)
- How to improve both instruction and data cache access, is it even possible
  - a much more thorough refinement of the data model and access patterns really needed

All: anything else I forgot and should be mentioned? We would need to separate a bit what we know almost certainly from speculative conclusions

Follow-up

- Reusable components and ideas for improving existing Geant4
  - Extending VecCore, VecMath, VecGeom
  - Investigate basketization of few performance-critical components in Geant4
- Follow-up plans
  - Compact specialized libraries as alternative to general stepping approach
  - Extraction of basketization generic library, possible basketization of FP-intensive components
  - Disentangling state from managers and move towards more functional programing style
    - Increase flexibility of functional-based regrouping, enable parallelism opportunities below event level.
  - Review the data model and flow in Geant4 to pre-empt extra parallelism and acceleration opportunities
  - …
- Pere will take care of most of this in his presentation

Conclusions

- Main messages we want to transmit
  - Innovative and disruptive R&D allowing to investigate novel technologies and approaches to simulation
  - Allowed to improve performance-critical code in the simulation chain
  - Allowed to develop new components, some being used now in production
  - Basketization failed to bring the expected gains in the full picture of the simulation, mainly due to data copy/management overheads.
    - Started from a wrong foot expecting to bring large overall gains in simulation
    - Does not mean it is not good for other flows (e.g. pipelines)
  - GeantV allowed to clear-up the interesting R&D paths and set the realistic expectations for the future
    - R&D work is not over, just gets more focused and specialized
- Pere, Daniel, Witek, Philippe, but also anyone else, I will gladly integrate conclusions that you deem important here