Performance results of the GeantV prototype with complete EM physics

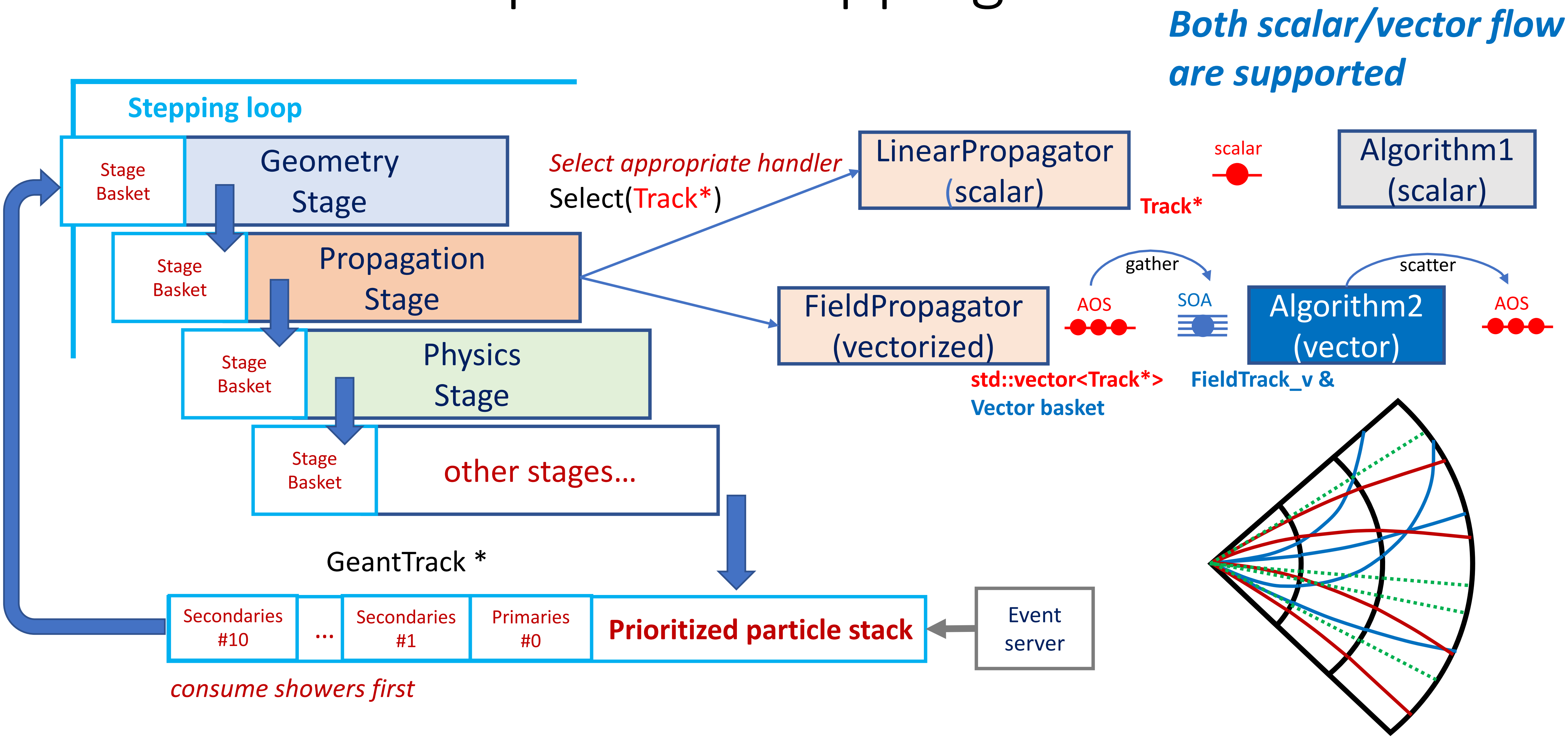
Andrei Gheata for the GeantV R&D team

ACAT 2019, Saas Fee

Vector Simulation R&D

- GeantV: performance study for a vector simulation workflow
  - An attempt to improve computation performance of Geant4
- Steering framework revisited
  - Track-level parallelism, “basket” workflow
  - Improving instruction and data locality, leverage vectorization
  - Adaptability to new hardware and accelerators
- Making simulation components more portable and vector friendly
  - VecGeom: modern geometry modeler handling single/multi particle queries
  - New physics framework, more simple and efficient
  - VecCore, VecMath: new SIMD API, SIMD-aware RNG and math algorithms

GeantV multi-particle stepping

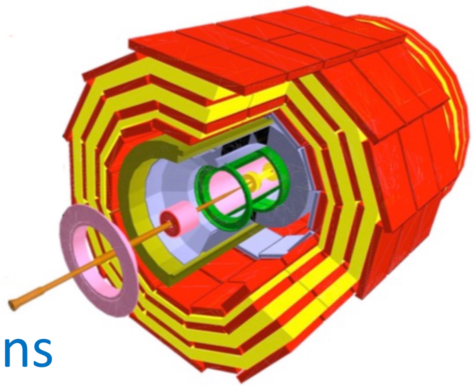


Where are we today?

- EM shower simulation
  - Detector model at full complexity of a LHC experiment
  - User interfaces integrated and tested by CMS (results @how2019)
  - First demonstrator for reproducibility (see talk of Soon Yung Jun)
- Ongoing performance study
  - Detailed comparisons: different GeantV modes and Geant4
- Preliminary set of conclusions including:
  - Vectorization and locality: benefits and limitations
  - Current limits of multi-threading in “basketizing” environments

What we compare

- Examples: simplified sampling calorimeter and a CMS simulation using 2018 geometry and 4T uniform field
  - Complete set of models for e^+, e^-, γ
  - Geant4 running equivalent physics list, field, geometry setup and cuts
  - Identical physics results, and equivalent #steps, energy deposits, particle yields
- GeantV: several configurations
  - Field ON/OFF (uniform field, field map version not yet efficient)
  - MT performance
  - Single track mode (emulating Geant4 tracking) -> locality
  - "Basketization" ON/OFF for different components -> vectorization gains
  - Vector baskets dispatched to scalar code -> measure overheads

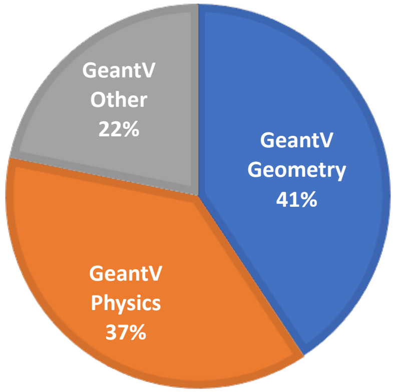


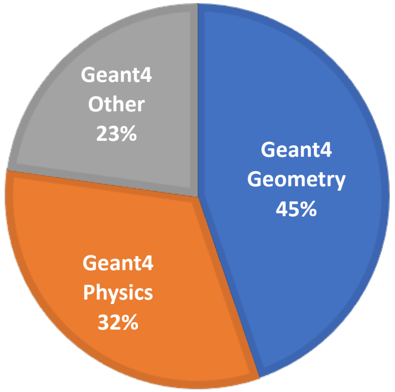
Preliminary performance: CMS example

- GeantV time performance improvement ranges from 1.9 to 2.1 depending on configurations (see backup slide 17)
  - Gains come from every component: geometry, physics, stepping management
  - Hard to disentangle component gains from a “background" of more efficient computation
  - The most efficient CMS GeantV configuration with a uniform field gives a factor of 1.92
  - The CMS experiment is working on realistic tests within the CMS simulation framework
- Global gains from vectorization and workflow can now be evaluated
  - Vectorization benefits: up to 15% total time
  - Basket workflow gains averaging at ~15% total time, with a large variance (0-30%) dependent on CPU architecture
- The rest of performance gain coming mostly from instruction locality
  - Analysis still ongoing, but performance counters showing far fewer instruction cache misses compared to Geant4

Component and global performance figures

- Similar time fractions by category, and very close number of FLOPS (GV/G4)
  - Geometry: important time reduction due to VecGeom navigation
  - Physics: more compact physics code
- Performance indicators better for GeantV
  - Computation intensity, CPU utilization
  - Far fewer instruction cache misses





	GeantV	Geant4	GeantV/ Geant4
FLOPS (DP_OPS)	1.86E12	1.67E12	**1.11**
FLOPS Per Cycle	0.26	0.13	2.00
Instructions Per Cycle	1.06	0.80	1.32
FLOPS per Memory Op	0.56	0.33	1.70
L1 instruction cache misses			1/7.7
L2 instruction cache misses			1/2.2
TLB misses			1/11.2

Intel(R) Xeon(R) CPU E5-2620 0 @ 2.00GHz

Vectorization performance: CMS example

- Fraction (% total CPU time) of code vectorized so far rather small
  - Physics: 7-11% final state sampling, 6-12% multiple scattering, 15-17% magnetic field propagation
  - Geometry: vectorized code in many branches (~4K volumes in CMS), not yet efficient to basketize
- Important intrinsic vectorization gain factors from unit tests
  - AVX2: Physics models: 1.3-2.5, geometry: 1.5-3.5, field propagation: ~2
- Visible vectorization gains in the total CPU time
  - Physics models: no gain (but MSC: 2-5%), geometry: performance loss, field propagation: 5-9%
  - Performance loss in case of "small" hotspots (e.g. geometry volumes)
- Basketizing is efficient only when applied to "dense FLOP" algorithms
  - Best basketized configuration in most recent tests brings ~10% (total CPU time) on Haswell AVX2 for vectorized code weighting ~35% (~1.4x visible speedup)

Locality from basketized workflow

- Hard to measure without comparing to equivalent stack-like approach
  - Implemented a special "single track" mode, transporting one track at a time through all stages (like Geant4)
  - Performance counters showing increased instruction cache misses, but less data cache misses
- Different levels of performance degradation in single track mode
  - Ranging from 0-30% depending on machine topology/simulation configuration: to be understood
- Only a small fraction of the performance improvement is due to basket workflow
  - Further analysis needed to disentangle all effects

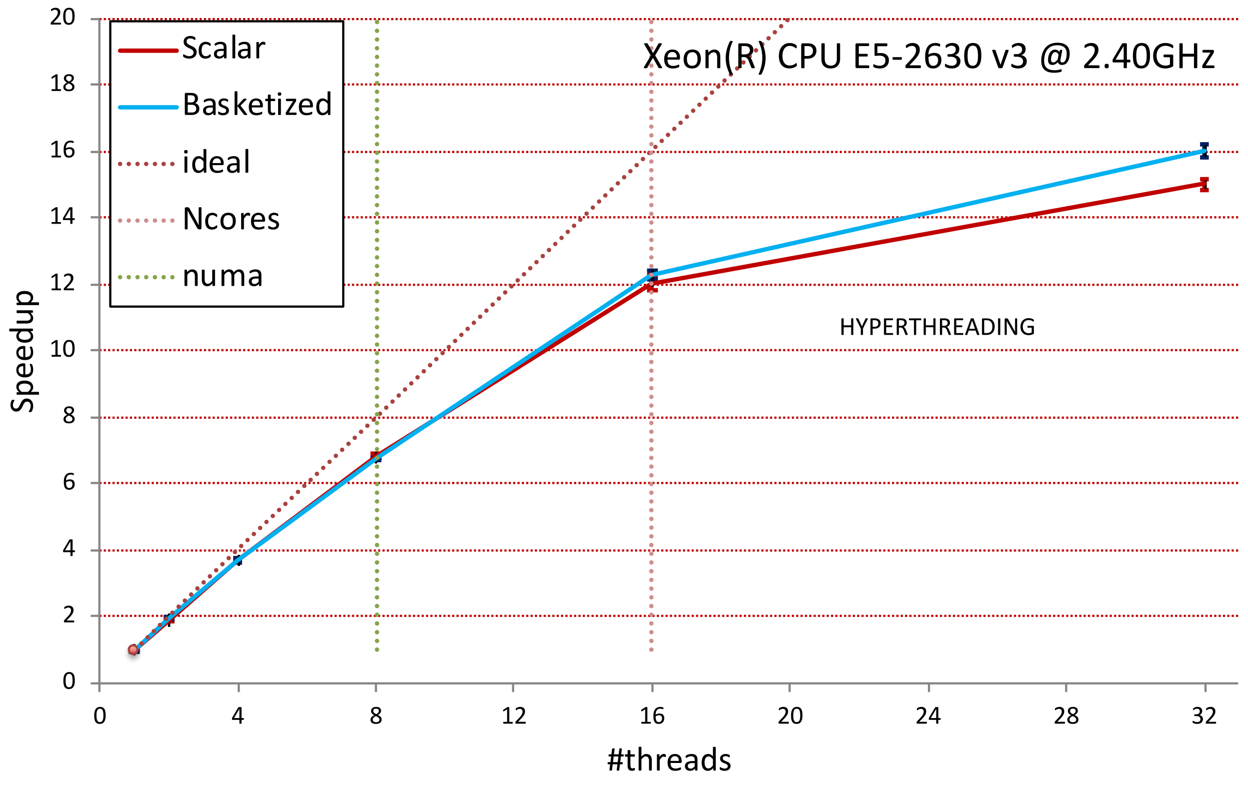
Preliminary conclusions for single thread performance

- GeantV uses fewer ‘clock cycles’ for the same number of FLOPs
- Better performance numbers overall: FPC, IPC, FPM
  - Fewer cache misses at several levels (specially L1 instructions, L2). Note that in basket mode instruction caches misses decrease, and data cache misses increase.
- The gains from workflow and vectorization explain only a small part of performance increase, what about the rest?
  - Simplified/more efficient code, library size, less deep call stack and less virtual calls – just some of the possible reasons
  - Quantifying these effects is very important
- The limits of applicability of the GeantV “basket” model now visible
  - Very hard to obtain vectorization benefits without reasonable hotspots

Multi-thread performance

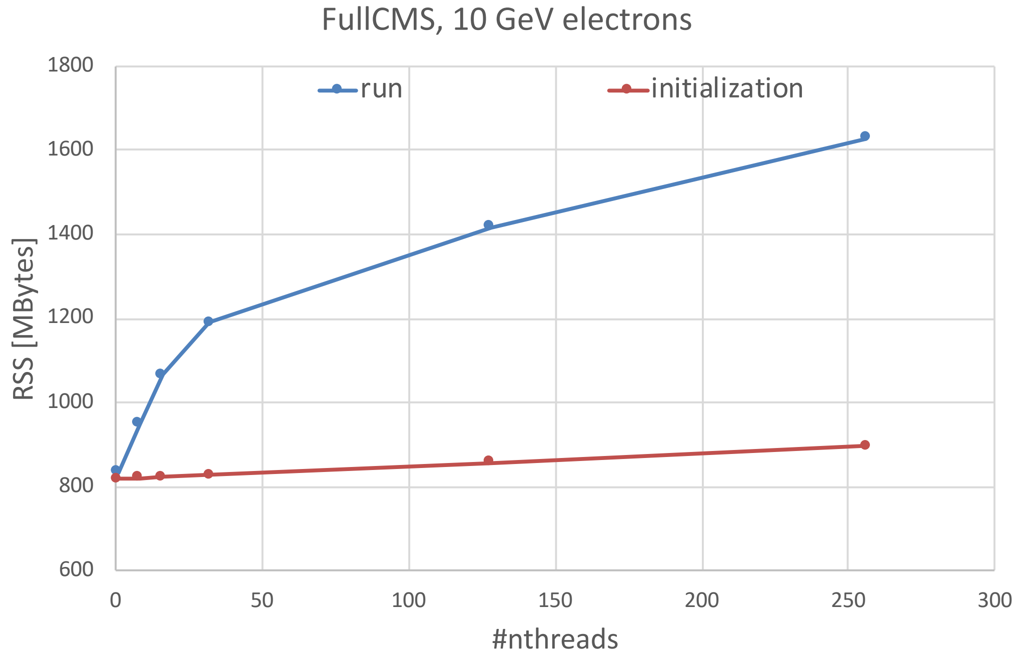
- Very different model compared to Geant4 MT
  - A pool of shared events in flight (GeantV) compared to one event per thread (Geant4)
- Sharing track workload among threads introduces overheads
  - Event tails introduce inefficiency, exacerbated by MT
  - The basket mode de-balances the work (winner takes all)
- Several improvements made to reduce serial part
  - Work stealing queues, memory contention reduced
- Will investigate reducing track sharing at the expense of more tracks in flight (more memory)
  - Sets of events (owned by/having affinity to) threads
  - May introduce tail problems

Current MT performance



Scalability for scalar and vector modes

Peak memory dependence on #threads, strong scaling, 10K electrons @10GeV



- Larger memory footprint than G4, but much more compact
- Code fitting L3 cache
- Data is pre-allocated in pools, producing less memory fragmentation than Geant4

Short-term work plan

- Deepen the performance analysis
  - Identify the cause of the bulk (60-70%) of the total gain, and the dependence on the architecture
  - Understand better differences compared to stack-like (Geant4) mode
  - Final fixes and consolidations for the beta release (now at pre-beta4)
- Understand the most profitable directions to work on to improve Geant4
  - Performance to be recovered by library restructuring (better fitting caches)
  - Code simplification: physics framework and step management
  - Better compromise between data and instruction locality, by adopting basket-like workflow in certain areas

Outlook

- GeantV vs Geant4 time performance improvement is ~1.9 for a standalone CMS application with a uniform magnetic field
  - CMS evaluating performance but also integration effort
- Contributions from basket workflow and vectorization do not explain the full gain, the major part is coming from improved instruction cache use
- Improvements for individual components visible but so far hard to disentangle from the profiling
- MT performance improved compared to previous versions, but still not ideal
  - The plan is to increase the event affinity to threads

Where do we go from here?

- The performance tag (beta) of GeantV demonstrator coming soon
  - Fixes and consolidations already available in a series of pre-beta tags
- Detailed performance benchmarking underway.
  - Conclusions are still preliminary
  - Short term plan for extending the analysis
- Finalizing this performance study will outline the directions to go
  - Technical document (facts, numbers and lessons learned) to be prepared
  - What are the directions for adopting some of these benefits in Geant4

Backup

Some preliminary performance numbers

- 4kgauss/nofield = simulation in constant field (4 Tesla) or no field
- Basketizing: physics (final state sampling), multiple scattering and field
- Counters shown below:
  - DP_OPS = Floating point operations; optimized to count scaled double precision vector operations
  - FPC = FLOPs per cycle
  - IPC = Instructions per cycle
  - FMO = FLOP's per memory operation
  - DCM, ICM = Data cache misses, instruction cache misses, shown as ratios

Intel(R) Xeon(R) CPU E5-2620 0 @ 2.00GHz, cache size : 15360 KB, MemTotal: 32 GB

	CPU time [s]	G4/GV	DP_OPS	FPC	IPC	FMO	TLB_DCM G4/GV	TLB_ICM G4/GV	L1_DCM G4/GV	L1_ICM G4/GV	L2_DCM G4/GV	L2_ICM G4/GV
GV-4kgauss	2722	**1.92**	1.86E12	0.26	1.06	0.56	0.74	11.16	1.38	7.63	0.55	2.24
G4-4kgauss	4987		1.67E12	0.13	0.8	0.33						
GV-nofield	1758	**2.10**		0.25	1.1	0.51	0.71	24.97	1.28	16.65	0.56	1.99
G4-nofield	3668			0.13	0.85	0.32						