HEP detector simulation in the 2030's

V. Daniel Elvira

Computational Science Seminar – ANL/FNAL/UChicago

September 24th, 2020

Outline

Detector simulation is of critical importance to the success of HEP experimental programs, a determinant factor for timely delivery of precise physics results[0]

- Introduction
  - HEP event, simulation workflow, tracks and showers, types of simulation
- The Geant4 simulation toolkit
  - The Collaboration, elements of a Geant4 simulation application
- Computing challenges in detector simulation
  - Simulation in numbers, resource gap, the brave new world of evolving computing technology
- Recent R&D activity
  - Results from the GeantV project
- Current R&D efforts
  - Adaptation for use of HPCs with accelerators, application of machine learning techniques
- The future
  - Outlook

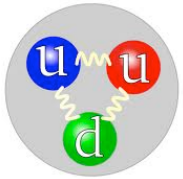
Introduction

HEP event, simulation workflow, tracks and showers, types of simulation

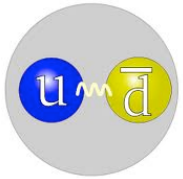
Particles and interactions

Protons, neutrons, electrons form the atoms

Hadrons (Baryons – protons, neutrons), mesons made of 2, 3 quarks):



Baryon
(i.e. proton)



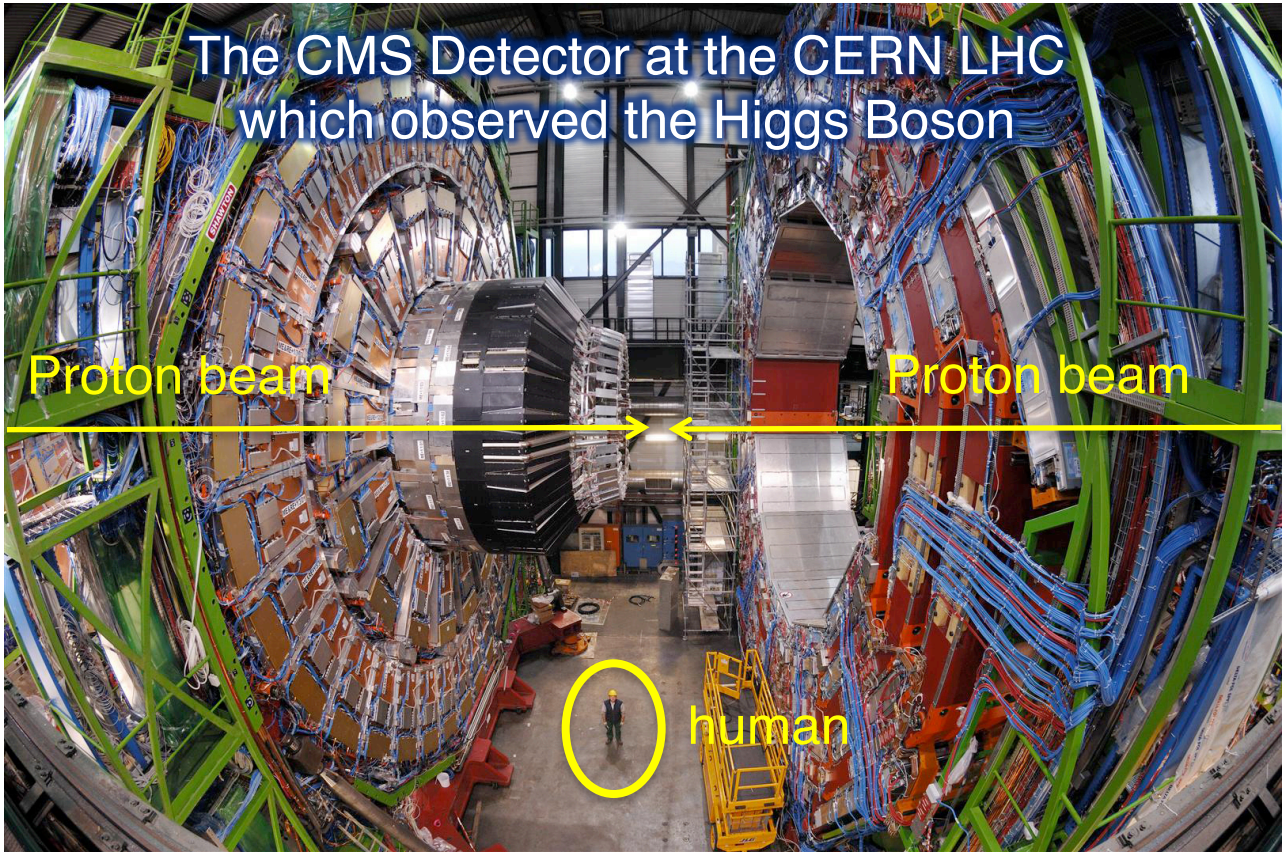
Meson
(i.e. pion)

Muons, pions, electrons, protons, neutrons, neutrinos, … are found in cosmic ray showers, and produced in particle accelerators

Particles with electric charge, photons, neutrinos ➔ EM interactions

Particles with quarks (and gluons) ➔ Nuclear interactions

High energy physics detectors



A high energy physics event (Example: CMS Experiment)


Higgs

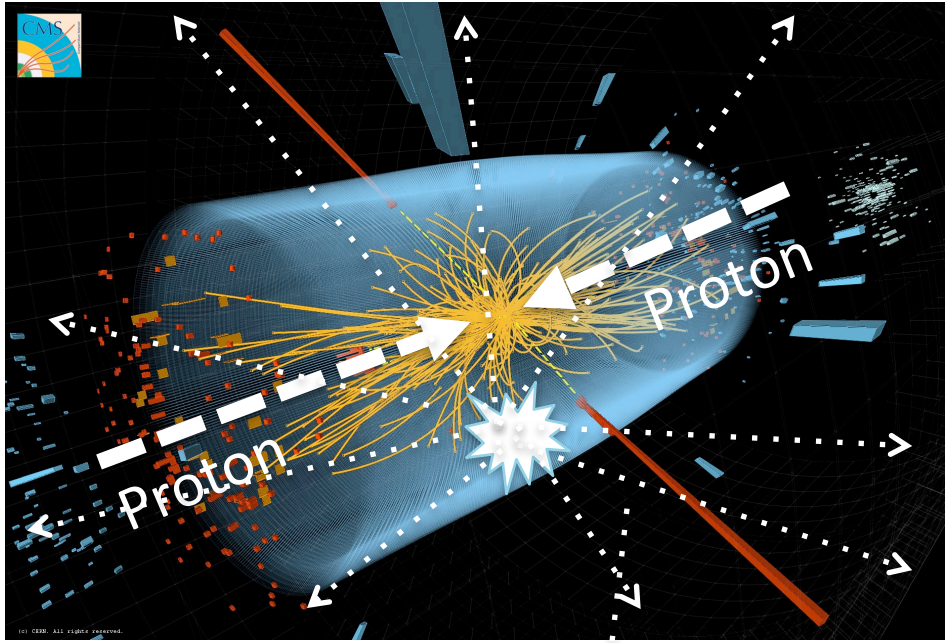

Quark jets


Gluon jets

Tens to hundreds of these “primary particles”


Photon


Dark Matter ?



Collider event:
all the detector data associated with a single pp collision


Z Boson

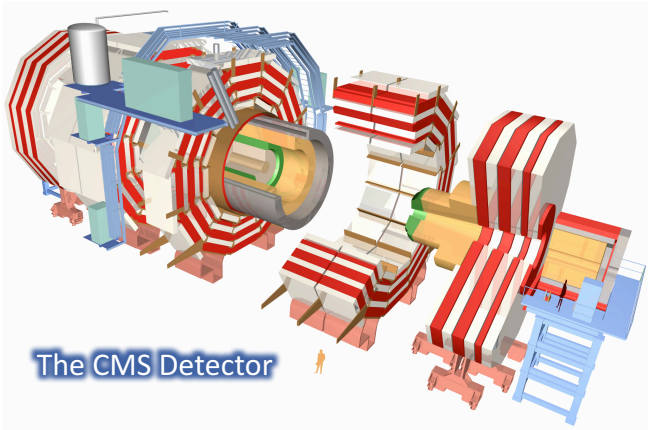

W Boson

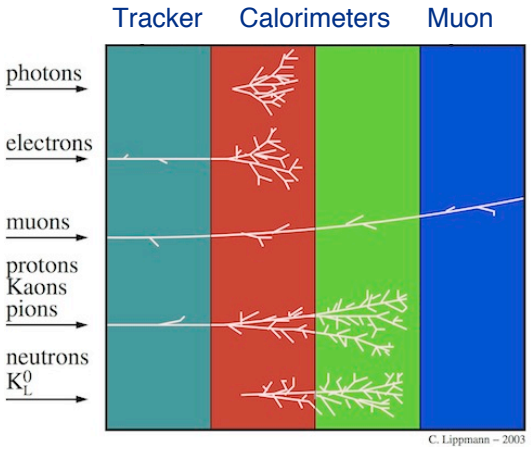

Electron


Muon


Tau

Particles through a collider detector: tracks and showers





Tracker measures charged particle trajectories

Calorimeter measures particle energy

Muon system, combined with tracker, measures muon trajectories

(Physics processes: energy loss, multiple scattering,…, etc. “Showers” of secondary particles produced through EM and nuclear interactions)

Hits and energy deposits in millions of detector channels ➔ **x**, **p**, E, time measurements

Particle tracks and particle showers must be modeled accurately

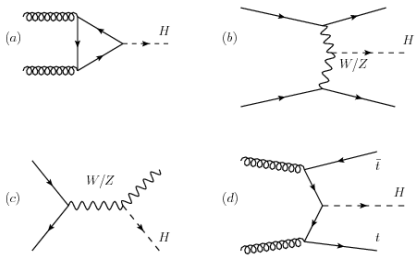
Why to simulate detectors

- Save time and money, improve the quality and accuracy of physics measurements

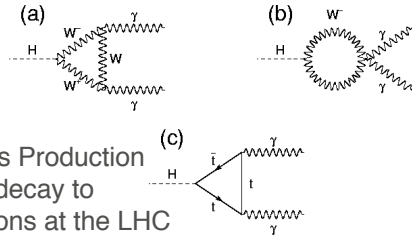
  Design optimal detector, best physics at a given cost, even before fastening the first screw!
- Simulation is not magic

  Particles cannot be “discovered” in a simulated sample which does not model them
- Simulation is essential to HEP experiments

  Teaches physicists what mark a new particle would leave in the detector if it existed

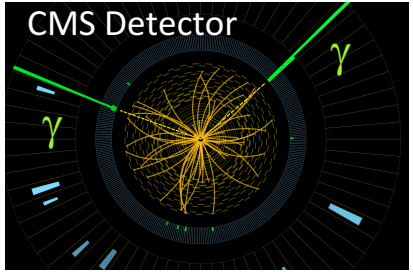


+



Higgs Production and decay to photons at the LHC

and

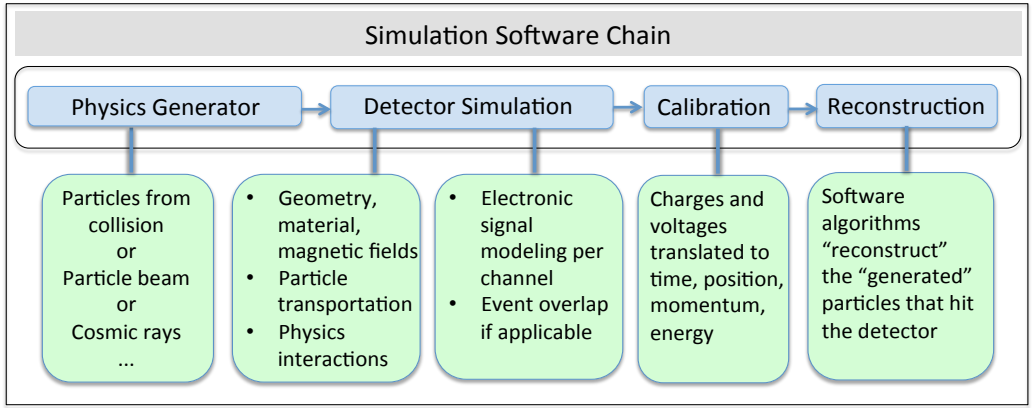


➔ Higgs discovered in July 2012

SM Higgs prediction:
Higgs is produced at the LHC and decays to two γ’s with given properties for the event and the individual particles

Observation:
two photon events with predicted detector marks are observed

Simulation software chain in a typical HEP experiment



Simulation referred to as "Monte Carlo (MC) simulation"

Simulated events referred to as "MC events, or MC samples"

- Physics generator: provides the final states of the physics process of interest (Pythia, Herwig, Madgraph, Alpgen, etc. in colliders; GENIE, etc. for neutrinos)
- Detector simulation [focus of this presentation]:
  - First stage: passage of generated particles through detector material and magnetic fields
  - Second stage: detector electronics, backgrounds to collision of interest (pileup)
- Calibration: from detector quantities to physics quantities
- Event reconstruction: algorithms, typically the same, applied to real data

Some history

Accurate computer simulation is essential to design, build, and commission the highly complex detectors in modern HEP experiments, and to analyze & interpret their data

- Old times detector simulation
  - Simple analytic calculations, back-of-the-envelope estimates
- Era of detailed detector simulation started in late 70's early 80's
  - Electron Gamma Shower (EGS[1]), GEometry ANd Tracking (GEANT) software
- GEANT3[2] software kit to describe complex geometry, propagate particles and model interactions as they traverse different materials and EM fields
  - GEANT3 widely used by CERN, DESY, FNAL experiments. First OPAL (LEP), then L3 and ALEPH, followed by experiments at DESY and FNAL in the 90's
- Other simulation tools are FLUKA[3] and MARS[4]
- Geant4[5,6] used by most HEP experiments – limited initially, the norm in 21st century

Types of simulation: toy, parametrized, full

- Toy simulation (ToySim) – a few simple analytical equations without a detailed geometry/field description or particle shower development
  - Zeroth order detector or physics studies
    - Output data format may not be the same as real data's , speed is a small fraction of a second/event
- Parametrized simulation (ParSim) – approximate geometry/field description, parametrized energy response and resolution, shower shapes
  - Computing intensive MC campaigns that would otherwise be prohibitive, i.e. parameter space scanning in BSM signal samples
    - Examples are the CDF QFL simulation (1990's) and CMS Fast Simulation framework which are tuned to test beam data, single tracks and/or full simulation
    - Output data format is typically identical to real data's, speed is of the order of a second/event

Types of simulation: toy, parametrized, full

- Full simulation (FullSim) – based on Geant, FLUKA, MARS with detailed geometry, field, and particle shower description
  - Detector and physics studies where geometry and physics accuracy are important
    - Output format same as real data's, speed is of the order of seconds to minutes per event

Full versus Fast simulation (FullSim vs. FastSim) – misleading concept

Experiments are moving towards simulation frameworks with flexibility to incorporate "fast simulation techniques" to a base Geant4 application

Tabulation, shower libraries, parametrization a la GFLASH, Machine Learning

The Geant4 simulation toolkit

The Collaboration, elements of a Geant4 simulation application

The Geant4 simulation toolkit Geant 4

At the core of most full simulation applications at modern collider experiments, i.e. LHC, is the Geant4 toolkit

- International Collaboration of tens of institutions and ~120 physicists and computer professionals, including members from CERN and FNAL/SLAC/LBNL/LLNL
- Written in OO C++, > 1 million lines of code, > 2000 C++ classes
- Used by almost all HEP experiments (10,000 users), space, and medical applications

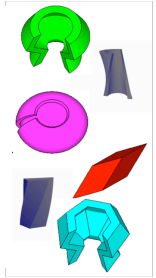
20th G4 Collab. meeting at FNAL, USA (2015)

24th G4 Collab. Meeting at Jeff. Lab., USA (2019)

A Geant4-based simulation application

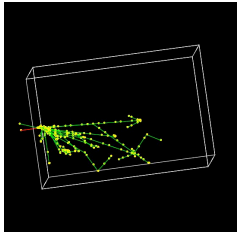
Experiments develop a ”simulation application” (software package) for their detector using Geant4 by assembling each of the following elements:

Detector geometry
(shapes and materials)



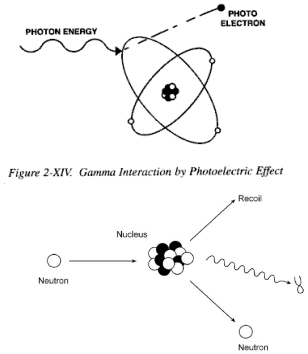
+

Particle Propagation through
geometry and EM fields



+

Physics Processes



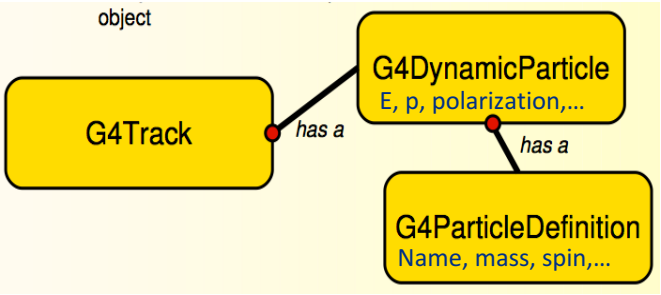
The user selects:

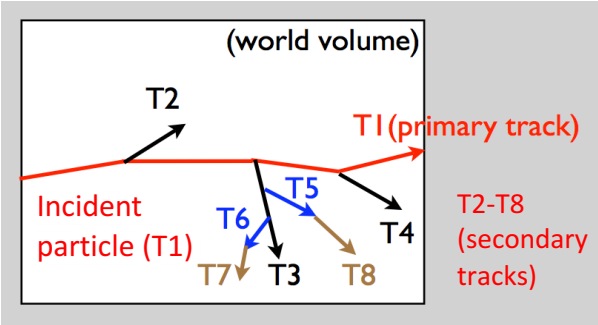
- Method of integration of the equation of motion, particle tracking parameters
- “Physics Lists” composed of a subset of the physics models available to describe the interaction of particles with matter for energy between 250 eV and ~100 TeV

Output is a collection of “particle trajectories” and “simulated hits” with position, time, and energy deposited in detector volumes

Geant4 transportation

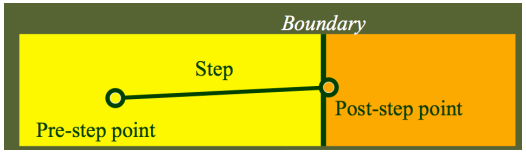
- In Geant4, a track or G4Track, primary (from a generator/beam) or secondary (decay product or shower component) has the info to transport particles through the detector





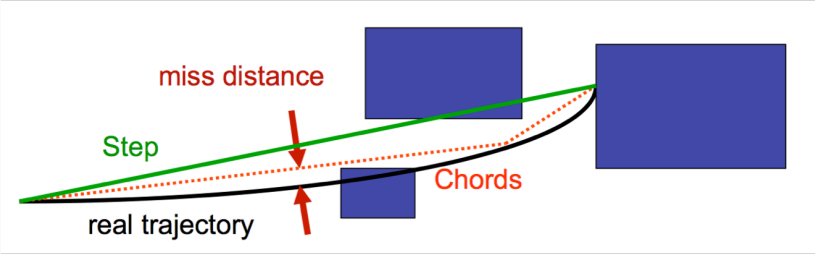
Tracking is sequential and follows "last in first out" rule: T1->T4->T3->T6->T7->T5->T8->T2

- The G4Track is updated after every G4Step, a step in the particle or track propagation
  - The user defines a maximum step length, steps also end when a physics process is invoked or at a volume boundaries (material or sub-detector transitions)



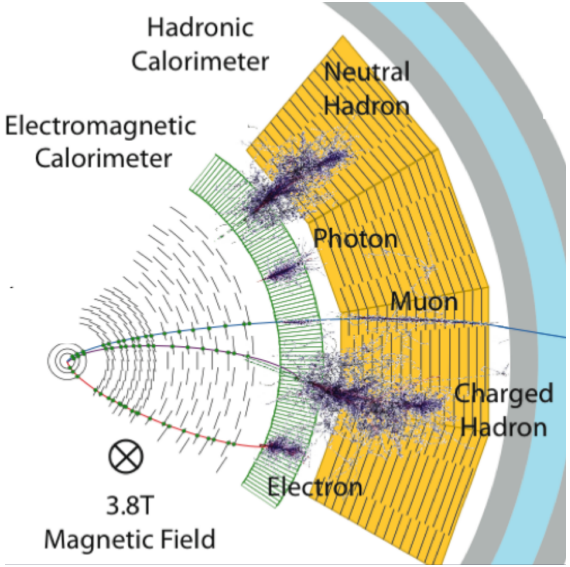
Geant4 magnetic field

- Particles propagated in EM fields by integration of equation of motion using the Runge-Kutta method (others also available)



- Curved path broken into linear chord segments to minimize the *sagitta* (maximum chord-trajectory distance)
  - Chords used to interrogate navigator on whether the track has crossed a volume boundary
  - miss distance parameter used to tune volume intersection accuracy

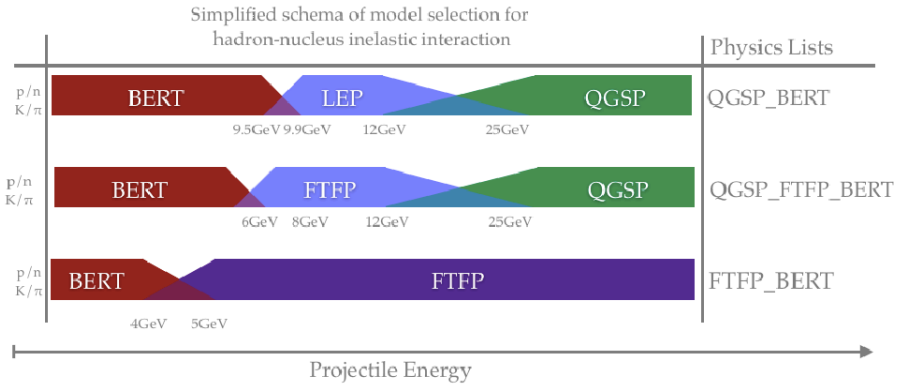
G4 supports uniform and non-uniform (static or time dependent) user defined magnetic fields

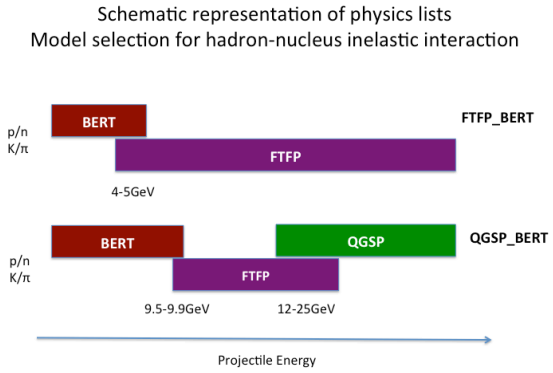


Particle Showers in CMS Detector

Geant4 physics

- Geant4 does not provide “one size fits all” modeling of physics. Instead:
  - Models for processes (cross sections, final state information for each particle type)
    - Categories: EM, hadronic decay, optical, photolepton-hadron, parametrization, transportation
- Production cuts on secondary particles set to a default 1 mm threshold
  - Secondary particles with $E_{kin} < E_{\text{needed to travel 1 mm}}$ are stopped
- Geant4 provides a set of “physics lists” or combination of models best suited to different application areas such as: HEP, LHC neutron fluxes, shielding, medical





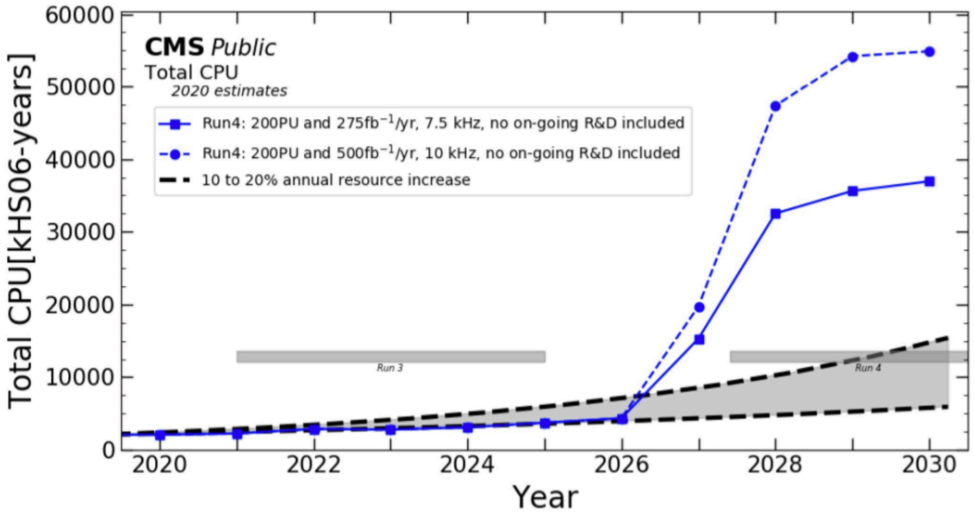
Computing challenges in detector simulation

Detector simulation in numbers, resource gap, the brave new world of evolving computing technology

HEP detector simulation in numbers

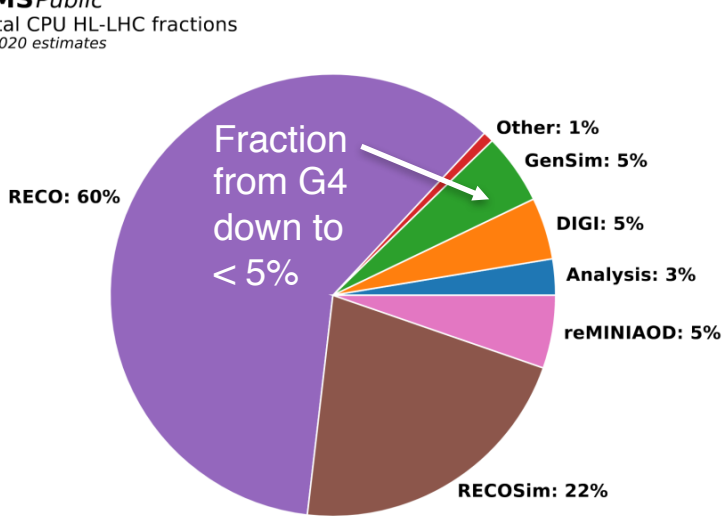
- 2010-2020: Each LHC experiment produced tens of billions of simulated events
  - Geant4-based simulation take seconds to minutes/event, depending on event physics
  - In Run 1 & 2, detector simulation was the largest consumer of computing resources
    - ATLAS/CMS: G4 module took ~40% of total, ATLAS's 8 times slower than CMS's
    - LHCb and ALICE used two orders of magnitude less CPU than ATLAS for G4 simulation
- HL-LHC will spent the largest fraction of computing resources in reconstruction
  - High luminosity means a 30 to 200 increase in hard interactions per crossing (pileup)
    - Reconstruction time grows exponentially with pileup, while G4 simulation dependence is flat
  - G4 simulation time will also increase significantly (e.g. x3/event in CMS – calorimeter granularity, more accurate physics)
    - However, it will decrease from 40% to < 5% of total resources in CMS, not so much in ATLAS
- DUNE will spend only a few percent of the total CPU cycles in G4 simulation
- G4 simulation brings moderate-to-high computing challenges
  - Moderate for some experiments (e.g. CMS) and high for others (e.g. ATLAS)
  - Depends on achieving Fast Sim with high physics fidelity (ATLAS's goal: 25% → 75% of samples)

HL-LHC computing resources gap – the CMS example



- Two scenarios: 7.5 KHz collected and 275 fb^{-1}/yr reconstructed, or 10 KHZ and 500 fb-^{1}/yr
- Computing resources increase of 10-20% /year
- Computing performance gains from R&D projects not included

CMS' projected CPU needs for 2030 are ~6x the available resources



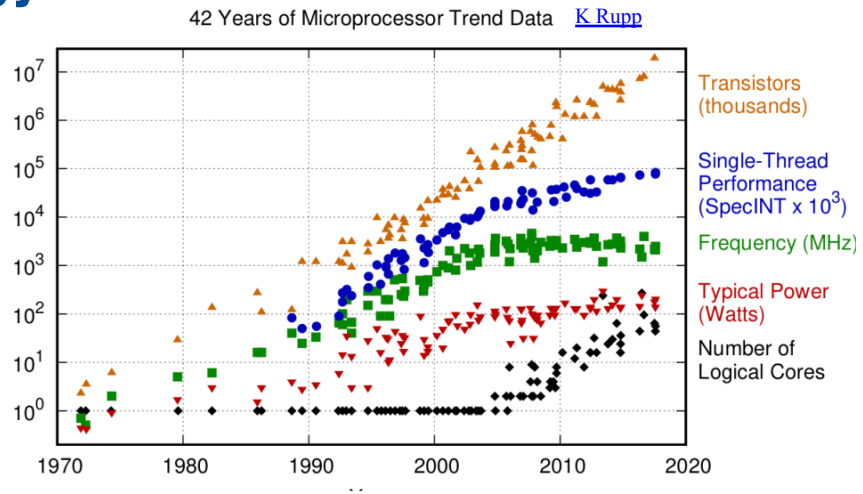
G4 simulation time per event 3x larger than in Run1-2, and larger MC samples will be needed to scale with luminosity

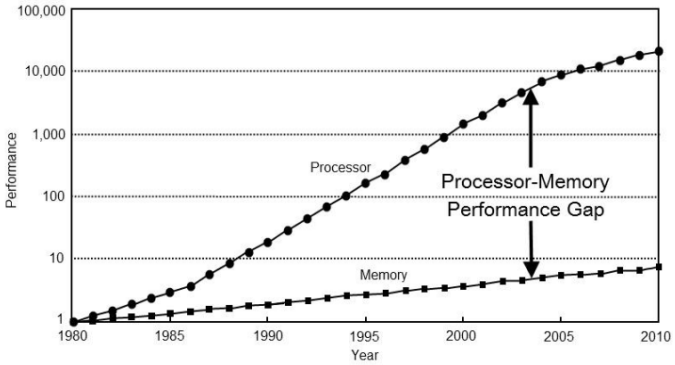
Evolution of computing technology

- Moore's law sort of holds (# transistors/vol.)
  - Doubling time is lengthening (> 2 years)
- Dennard scaling (dissipated power/volume independence with the # transistors)
  - Breaks down at ~3 GHz clock speeds (Power Wall driven by heat and cost)
- Processor and memory performance gap
  - CPU grows in speed and memory in size
  - Memory access times are now ~100s of clock cycles

We can not count on the performance of a CPU chip to double every 18 months

(Good assumption until ~2005)

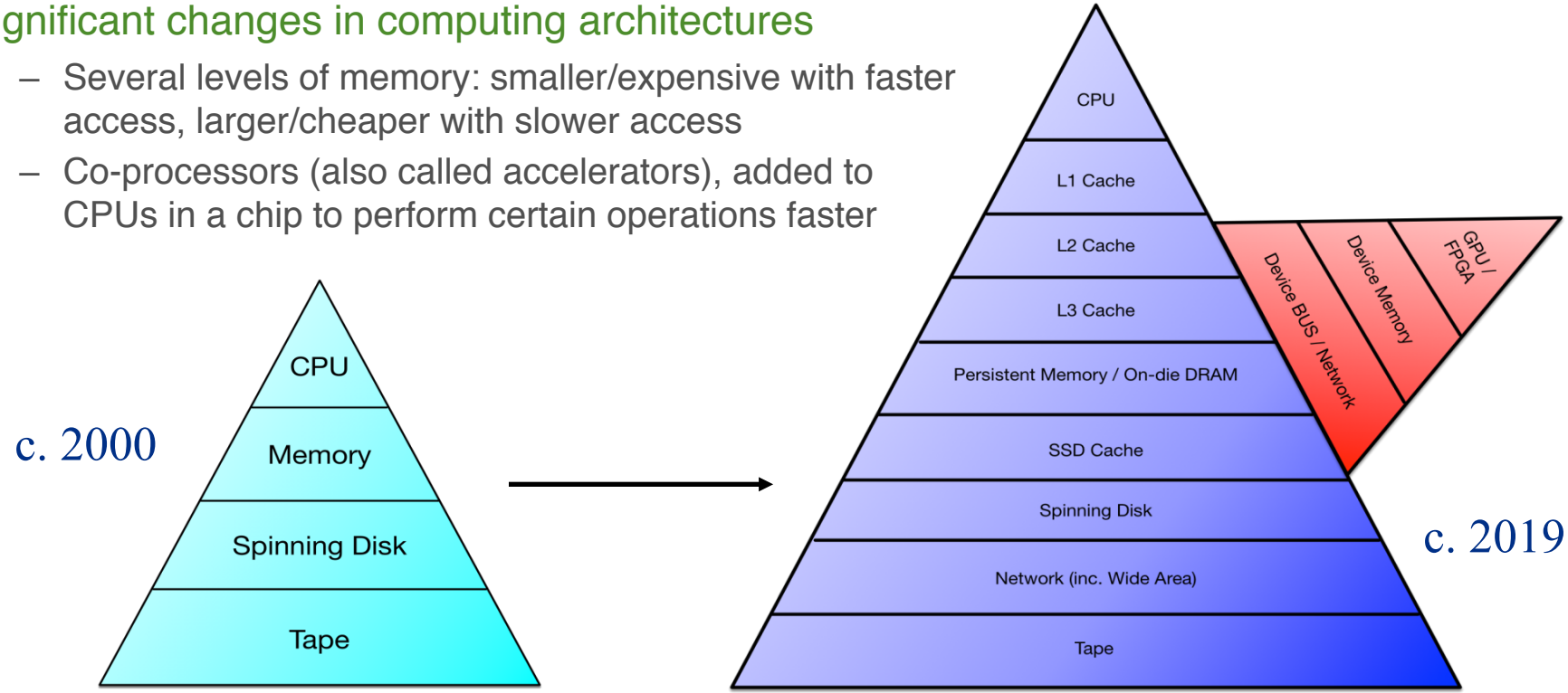




Evolution of computing technology

Significant changes in computing architectures

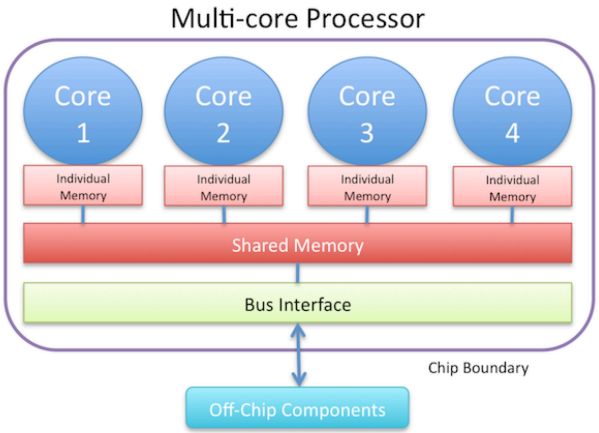
- Several levels of memory: smaller/expensive with faster access, larger/cheaper with slower access
- Co-processors (also called accelerators), added to CPUs in a chip to perform certain operations faster



The brave new world

Solution for the shortage in HEP computing resources will have to leverage growth in:

- Core count (multi-core machines)
  - Concurrency and parallelism
  - Multithreading, vectorization – SIMD
  - *Amdhal Law* limit on speedup is 1/(1-p) with p parallelizable fraction of code
- Co-processors also called accelerators
  - GPU: matrix operations, FPGA: configurable through software, ASIC: application specific
- Artificial intelligence techniques
  - Based on software algorithms: training, inference
  - TPU: ASIC for AI



NVIDIA's GPU

Altera's FPGA

Google's TPU

A community effort

- R&D to address S&C challenges in the next decade must be a community effort
  - LHC experiments, neutrino, muon experiments face similar challenges
  - Recourses are limited, cannot afford duplicated efforts
- The HEP Software Foundation (HSF) was formed in 2015
  - Followed early incarnations starting with a “Workshop on Concurrency in the many-Cores Era” (FNAL, 2011), and “Annual Concurrency Forum Meetings” (FNAL-2013, CERN-2014)
  - HSF facilitates coordination and common efforts in S&C across HEP in general
  - No formal organization, no representation from funding agencies, research institutions or experiments
  - A “coordination team” composed of volunteers contribute their time
- The community white paper
  - Year-long process, many working groups, two major workshops
  - Massive community engagement: 310 authors from 124 institutes, 14 chapters
  - **Published in Comp. and Soft. for Big Sci.:** https://doi.org/10.1007/s41781-018-0018-8
  - **Dedicated white paper on detector simulation:** https://arxiv.org/abs/1803.04165

Recent R&D activity

Results from the GeantV project

The GeantV R&D project

The GeantV prototype was designed to take the opportunities offered by modern computing architectures (2013-2019)

Paradigm shift – Particle-level parallelization (not event-level as G4). One thread may process particles from different events

Use parallelization techniques

- Instruction pipelining → parallel instruction handling within a processor
- Data locality and explicit vectorization
  - Single Instruction Multiple Data (SIMD)
  - Work offloading to accelerator

GeantV

Results from the prototype of concurrent vector particle transport simulation in HEP

G. Amadio[a], A. Ananya[a], J. Apostolakis[a], M. Bandieramonte[a,b], S. Banerjee[c], A. Bhattacharyya[d], C. Bianchini[c,f], G. Bitzes[a], P. Canal[c], F. Carminati[a], O. Chaparro-Amaro[g], G. Cosmo[a], J. C De Fine Licht[a], V. Drogan[a,h] L. Duhem[j], D. Elvira[c], J. Fuentes[k] A. Gheata[a], M. Gheata[a,l], M. Gravey[m], I. Goulas[a], F. Hariri[a], S. Y. Jun[c], D. Konstantinov[a,u], H. Kumawat[d], J. G. Lima[c], A. Maldonado-Romo[g], J. Martínez-Castro[g], P. Mato[a], T. Nikitina[a,p], S. Novaes[e], M. Novak[a], K. Pedro[c], W. Pokorski[a], A. Ribon[a], R. Schmitz[s], R. Seghal[d], O. Shadura[a,r], E. Tcherniaev[a], S. Vallecorsa[a,p], S. Wenzel[a], Y. Zhang[a,t]

Received: date / Accepted: date

Abstract Full detector simulation was among the largest CPU consumer in all CERN experiment software stacks for the first two runs of the Large Hadron Collider (LHC). In the early 2010s, it was projected that simulation demands would scale linearly with increasing luminosity, with only partial compensation from increasing computing resources. The extension of fast simulation approaches to cover more use cases that represent a larger fraction of the simulation budget is only part of the solution, because of intrinsic precision limitations. The remainder corresponds to speeding up the simulation software by several factors, which is not achievable by just applying simple optimizations to the current code base. In this context, the GeantV R&D project was launched, aiming to redesign the legacy particle transport code in order to benefit from features of fine-grained parallelism, including vectorization and increased locality of both instruction and data. This paper provides an extensive presentation of the results and achievements of this R&D project, as well as the conclusions and lessons learned from the beta version prototype.

Keywords Detector Simulation, Particle Transport, Concurrency, Parallelism, Vectorization

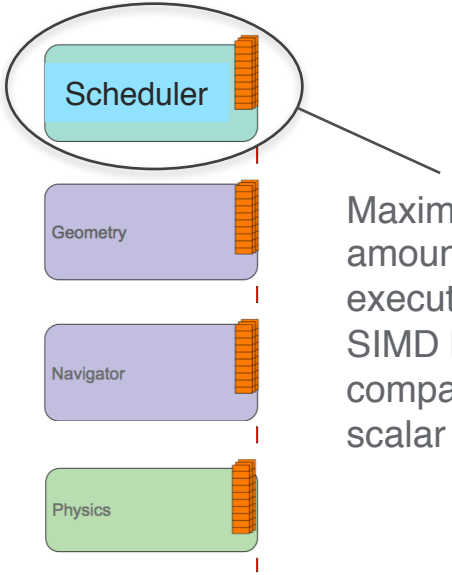
CERN, FNAL, UNESP (Brazil) BARC (India)

~5-12 FTE/yr for 7 years

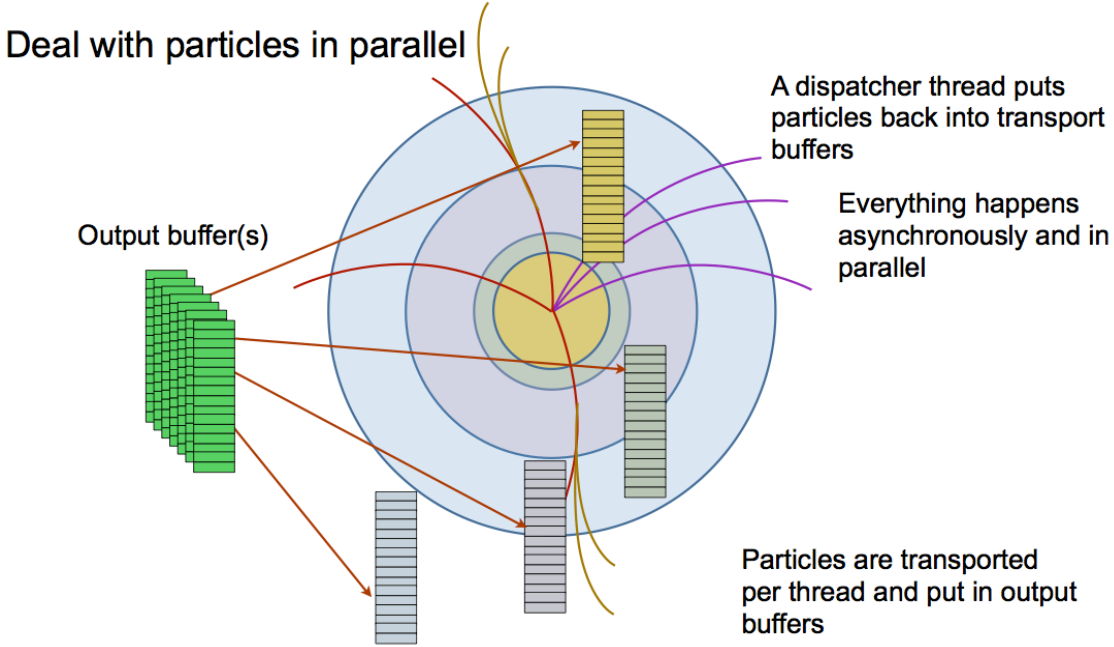
Paper[7] with results and recommendations accepted for publication in Computing and Software for Big Science – discussed next

GeantV basketized transport

Particles grouped in baskets by common characteristics: volume traversed, physics process, particle type



Maximizes the amount of work executed via SIMD baskets compared to scalar mode



Instruction pipelining

Available in most processors

Instruction processing not sequential but *concurrent* and in *parallel*

Instr. No.	Pipeline Stage						
1	IF	ID	EX	MEM	WB		
2		IF	ID	EX	MEM	WB	
3			IF	ID	EX	MEM	WB
4				IF	ID	EX	MEM
5					IF	ID	EX
Clock Cycle	1	2	3	4	5	6	7

IF: instruction fetching
ID: instruction decoding
EX: instruction execution
MEM: memory access
WB: register write back

(programmer must avoid conditional branching!)

Newer hardware architectures utilize instruction pipelining in an increasingly more efficient way

Vectorization and SIMD

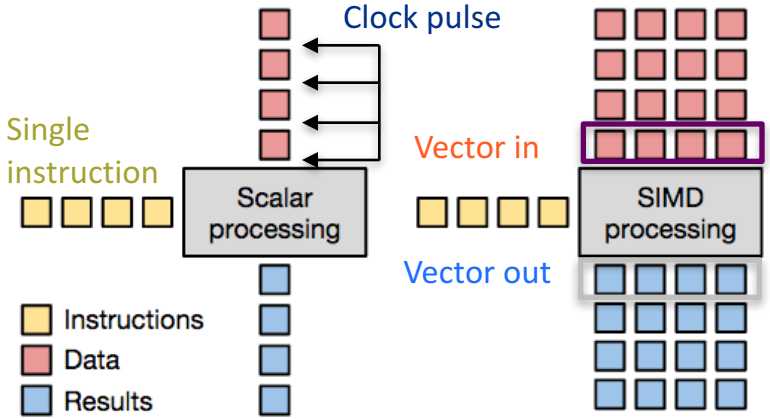
Vectorization for Single Instruction Multiple Data (SIMD) processing

- Code *vectorization* consists of organizing data in vectors to be stored simultaneously in processor *registers*
- Example: a single (same) instruction may be applied in one clock cycle to track data from different events
  - Apply same single instruction to a vector of data (SIMD)

Compilers have options for auto-vectorization, i.e.

```
g++ -mavx   or   -msse4.2
```

But …



Vectorization and SIMD

Vectorization for Single Instruction Multiple Data (SIMD) processing

Compilers can organize data in vectors:

```
// Example 12.1a. Automatic vectorization
const int size = 1024;
int a[size], b[size];
// ...
for (int i = 0; i < size; i++) {
   a[i] = b[i] + 2;
}
```

but cannot deal with branches:

```
// Example 12.4a. Loop with branch

// Loop with branch
void SelectAddMul(short int aa[], short int bb[], short int cc[]) {

   for (int i = 0; i < 256; i++) {
      aa[i] = (bb[i] > 0) ? (cc[i] + 2) : (bb[i] * cc[i]);
   }
}
```

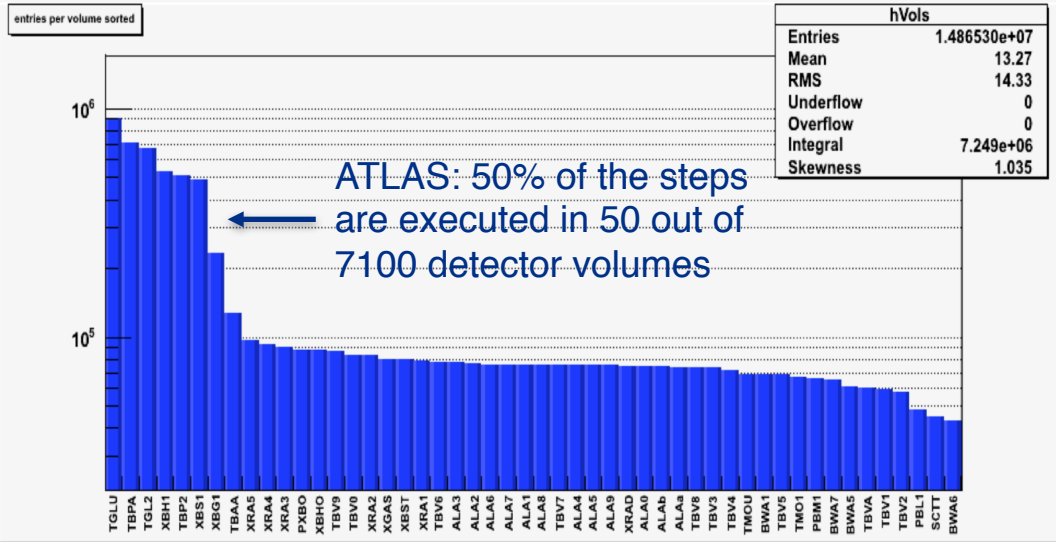
Vectorization by hand:

```
// Branch/loop function vectorized:
void SelectAddMul(short int aa[], short int bb[], short int cc[]) {

   // Make a vector of (0,0,0,0,0,0,0,0)
   __m128i zero = _mm_set1_epi16(0);
   // Make a vector of (2,2,2,2,2,2,2,2)
   __m128i two  = _mm_set1_epi16(2);

   // Roll out loop by eight to fit the eight-element vectors:
   for (int i = 0; i < 256; i += 8) {
      // Load eight consecutive elements from bb into vector b:
      __m128i b = LoadVector(bb + i);
      // Load eight consecutive elements from cc into vector c:
      __m128i c = LoadVector(cc + i);
      // Add 2 to each element in vector c
      __m128i c2 = _mm_add_epi16(c, two);
      // Multiply b and c
      __m128i bc = _mm_mullo_epi16 (b, c);
      // Compare each element in b to 0 and generate a bit-mask:
      __m128i mask = _mm_cmpgt_epi16(b, zero);
      // AND each element in vector c2 with the bit-mask:
      c2 = _mm_and_si128(c2, mask);
      // AND each element in vector bc with the inverted bit-mask:
      bc = _mm_andnot_si128(mask, bc);
      // OR the results of the two AND operations:
      __m128i a = _mm_or_si128(c2, bc);
      // Store the result vector in eight consecutive elements in aa:
      StoreVector(aa + i, a);
   }
}
```

Vectorization applied to geometry, magnetic field, EM physics, …

Data locality and SIMD

Baskets of particles of same type, same volume traversed, or similar kinematic properties → same set of geometry, field, or physics instructions



Locality not exploited by the classical transport

Basket data and instructions local to processor executing thread

GeantV performance results

Different aspects of GeantV performance were examined:

1. *Intrinsic* performance for GeantV applications on different architectures, compilers, including G4/GV CPU performance comparisons for equivalent applications

Table 6 Properties of the hardware platforms used for performance tests: CPU [GHz], Memory [GB] and L3 Cache size [MB].

Processor	CPU	Memory	L3 Cache
Intel E2620 (Sandy Br.)	2.0	32	15
Intel E2680 (Broadwell)	2.4	128	35
AMD 6128 (Opteron)	2.3	64	12

Table 7 Performance comparison between GeantV and Geant4: the average CPU time in seconds per event for simulating sixteen 10 GeV electrons propagating through the CMS detector and the magnetic field.

Processor	SIMD	Geant4	GeantV	Speedup
Intel E2620	AVX	4.94	2.33	2.12
Intel E2680	AVX2	2.18	1.63	1.43
AMD 6128	SSE4	6.63	4.33	1.53

G4/GV (CPU) ~1.45-2.1

Table 8 Performance comparison between GeantV and Geant4: the impact of different configurations of the magnetic field on Intel E2620 (Sandy Bridge). In the CPU performance ratios G4/GV and G4/GV(vect), the denominators refer to GeantV in scalar mode and vector mode, respectively.

Configuration	GeantV [s]	G4/GV	G4/GV(vect)
Zero field	1794	1.86	1.95
Uniform (3.8T)	2412	1.97	2.19
CMS field map	2621	1.88	2.12

G4/GV (CPU) ~1.95-2.2

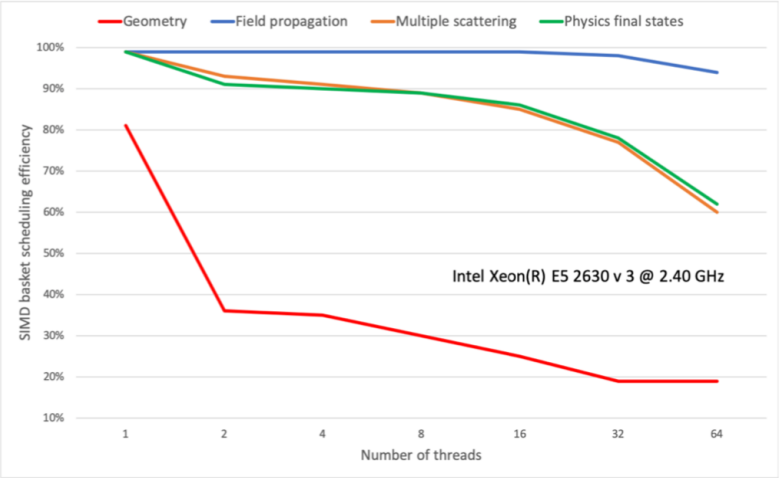
- Table 6: properties of the *hardware platforms* used for performance tests
- Table 7: *G4/GV CPU performance ratio* comparison for CMS standalone application
- Table 8: *impact of magnetic field complexity* on CMS G4/GV CPU performance ratio

GeantV performance results

Different aspects of GeantV performance were examined:

2. *Workload scheduler* performance while varying several parameters
   - Number of events in flight, basket size, scalar versus basketized mode

- *Field propagation, multiple scattering physics* are popular baskets → *high SIMD efficiency*
- *Geometry baskets* activated on a high track multiplicity watermark → *low SIMD efficiency*
- Scheduler fires *partially filled baskets in scalar mode* when available tracks are exhausted → *reducing SIMD efficiency*



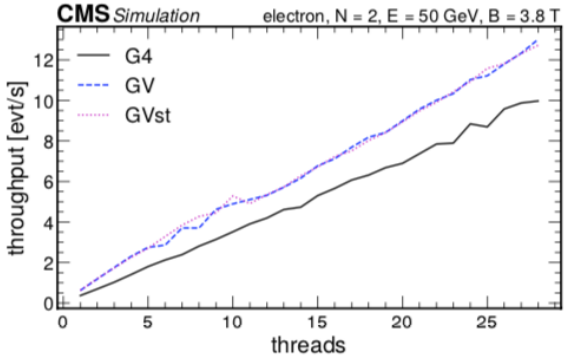
Multi-threaded mode: SIMD baskets filed concurrently to maximize population per category → buffered tracks consumed faster, triggering frequent scalar executions

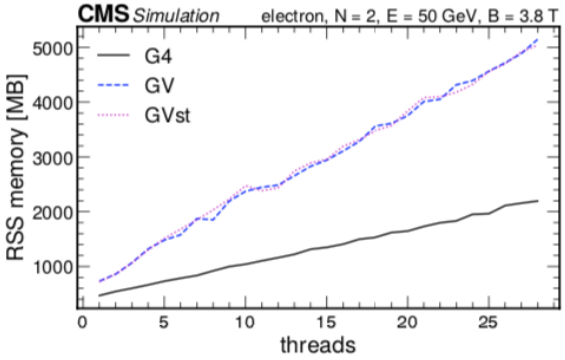
GeantV performance results within CMSSW framework

Different aspects of performance were examined:

3. GeantV performance within a *real experiment framework* (e.g. CMS' CMSSW)
   - 500 events with two electrons, E=50 GeV, B=3.8 T uniform field

Machine	Clock [GHz]	Cache [kB]	Cores	Throughput [GV/G4] Standalone 1 thread	Throughput [GV/G4] CMSSW 1 thread	Throughput [GV/G4] CMSSW N threads	RSS memory [GV/G4] CMSSW 1 thread	RSS memory [GV/G4] CMSSW N threads
E5-2683 v3	2.00	35840	28	1.60	1.69	1.30	1.56	2.34
Gold 6248	2.50	28160	20	1.49	1.66	1.18	1.54	2.31
E5-2660 v2	2.20	4096	8	2.14	2.63	2.18	1.42	2.36





CMSSW speedup exceeds standalone speedup

(smaller GV instruction size allows more CMSSW instructions to be cached by CPU)

GV/G4 throughput ~ 1.7-2.6

Speedup (memory) decreases (increases linearly) with # of threads

(GeantV uses more memory)

Outcome from GeantV R&D: lessons learned

- GV and G4 applications are modular and granular – no unexpected bottlenecks
  - Top function in Geant4/V take 6.5-1.6% of the total CPU time
- Trade-off between memory use and efficiency is a challenge
  - Restriction to the number of geometry shapes for which tracks were collected
  - Number of events in flight had to be limited → basket starvation and costly event closing
- Load balancing between multiple threads is another challenge
  - Thread sharing limited to threads pinned to a NUMA domain →reduced vector efficiency
- GeantV's much simpler code structure and smaller libraries results in far fewer instruction cache misses and is at the core of the overall performance speedup

Processor	ICM		DCM	
	GV	G4	GV	G4
Intel E2620	19	36	86	46
Intel E2680	23	29	101	51
AMD 6128	17	3.6	55	10

L1 cache misses

Processor	ICM		DCM	
	GV	G4	GV	G4
Intel E2620	54	429	218	269
Intel E2680	39	511	188	272
AMD 6128	49	309	141	144

L2 cache misses

Outcome from GeantV R&D: lessons learned

- A relatively small percentage of the code was vectorized (geometry, magnetic field, most EM physics) but relative gain with respect to scalar code is small
  - Vector instructions in scalar mode come from auto-vectorization
  - Poor vector performance in geometry due to the execution of sequential algorithms in navigation
  - Basketization overheads in the range of 10-25%, resulting in reduction of CPU speedup

Mode	PAPI_DP_OPS	PAPI_DP_VEC	%	Gain
Scalar	1770	277	15.67	-
Geom-vec	1771	333	18.82	0.96
Field-vec	1858	814	43.83	1.08
MSC-vec	1789	397	22.24	1.02
Phys-vec	1785	343	19.25	1.00
All-vec	1868	1051	56.26	1.00
Opt-vec	1868	996	53.35	1.12

- Integration of the GeantV prototype to a real experiment framework (CMSSW) was relatively straightforward and the run-time performance was the same or better than in the standalone example
  - The co-development model followed between GV and CMSSW developers was essential
    - Several iterations adapting the GeantV scheduler, CMSSW external work features, and the interfaces – never develop a simulation toolkit without talking to the experiments since inception

Outcome from GeantV R&D: products delivered

Many of the products delivered by the GeantV R&D project were integrated to the Geant4 simulation toolkit resulting in improved physics and computing performance

- *Simpler code with smaller libraries*
  - To be used in modernization of Geant4 for improved computing performance
- *VecGeom vectorized geometry package which also runs in GPUs*
  - At the core of current R&D for Geant4 GPU adaptation
  - Provided a 7-14% speedup to the CMS Geant4 application
- *VecCore framework*
  - For abstracting vector operations to run on heterogeneous architectures
  - Improved run-time performance of algorithms with both vector and scalar input
- *EM physics ($e^{\pm}/\gamma$ interactions)*
  - Most intensively used and computationally demanding
  - Physics algorithms reviewed for improved physics and computing performance

Current R&D projects

Adaptation of Geant4 for effective use of HPCs with accelerators, application of machine learning techniques to detector simulation

A detector simulation R&D program

Community decision on GeantV: cost of completing the project outweighed the benefits of the basket/vector approach – GV code is the base of much of current and future R&D

- The case for detector simulation R&D is still strong:
  - Detector simulation toolkits, such as Geant4, are more that 25 years old and were designed for sequential programing and homogeneous computing on CPUs
    - GeantV work uncovered significant opportunity for code simplification, optimization, factorization
    - Supercomputing facilities available to HEP require effective use of their resources
    - Simulation code may not even run on commercial hardware in the 2030s if not adapted

- A detector simulation R&D program must be comprehensive and flexible including:
  - Toolkit modifications to take opportunities offered by heterogeneous computing architectures
  - Application of AI techniques for high-fidelity fast simulation
  - Adaptation of full/fast simulation workflows for effective/efficient use of HPCs with accelerators
  - Portability solutions to keep up with rapid evolution of computer hardware

Celeritas – a GPU-based particle transport for HEP simulation

Partnership between the Exascale Computing Project (ECP) and the Computational HEP US Department of Energy programs: ANL, FNAL, ONL

Celeritas[8] would be an adaptation of Geant4 to have GPU accelerated transport

- Not a replacement of G4 but will include a mechanism to incorporate GPU acceleration
- Utilize leadership class hardware (GPUs)
- Support a complete set of physics models required by HEP experiments
- Leverage recent R&D products such as the VecGeom GPU implementation

Demonstrator consists of the GPU implementation of a transport loop

- Transport step loop on GPU including EM field, secondaries, scoring in user-selected cells
- EM physics for γ and $e^{\pm}$, hadron proxy for shower performance testing (first year)
- Performance testing, Figure of Merit (FOM) measurements on Summit/Volta GPU hardware
  - FOM = N_{evts}/time over full machine resource

AdePT – accelerated demonstrator of EM particle transport

R&D project started by the CERN Software group to demonstrate a realistic complete simulation workflow on GPU

The AdePT prototype[9] goals in a 6 months to 1 year timescale are:

- Understand opportunities and limitations for GPU usage in full HEP simulation
- Evaluate feasibility, effort needed, and performance expectations for a full HEP application that offloads EM shower simulation to GPUs
- Leverage existing experience and products including VecCore/VecGeom, ALICE GPU reconstruction, LHCb Allen framework, performance portability libraries

Demonstrator consists of a GPU transport engine

- Dynamic kernel scheduling, management of workflow and state data
- Adapt, develop, or optimize GPU-friendly transport components
- Understand constrains/hard limits and find solutions for data handling, memory management, kernel scheduling, GPU performance

A framework for optical photon propagation in GPUs

R&D project undertaken by the FNAL simulation group to accelerate γ propagation modeling in Liquid Argon G4-based simulation applications for neutrino and dark matter experiments

Motivation for the ***Optical Photons Framework***[10]:

- LAr TPCs collect signal from e's/optical γ's drifting/propagating to a readout plane
- G4 CPU applications take hours to simulate one event in Time Projection Chambers (TPC)
  - Experiments use tabulation at the cost of reduced physics accuracy
- There is an ***Opticks package***[11] (Simon Blyth) that integrates GPU ray tracing with Geant4 and reduces the execution time by orders of magnitude

The Optical Photons Framework will:

- Enable heterogeneous computing through the G4Tasking feature (J. Madsen – LBNL)
  - Option to propagate G4 optical photons in either CPUs or GPUs
- Allow execution of G4 Cerenkov, scintillation and wavelength shifting processes in GPUs
- Provide sensitive detector plugins for different detector types, persistent hit collections, timing and memory information

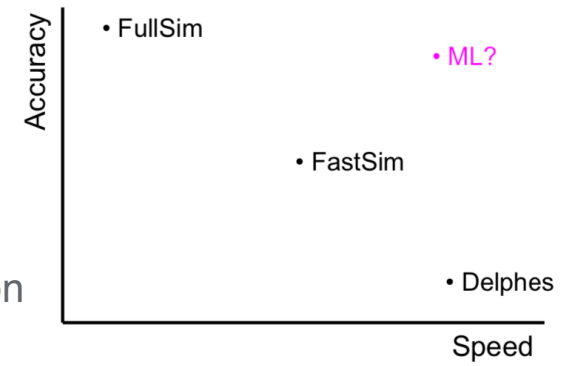
Machine learning for Simulation

Parameterized simulation is ~100x faster than G4 and models physics to within ~10%

- Reason why ATLAS processes < 25% of MC with ParSim and CMS only signal samples

- ML algorithms can be used to generate or de-noise simulation output (typically calorimeter showers using GANs or CNNs - significant R&D activity in progress)
  - ***Pros:*** may achieve higher accuracy than parametrized simulation, faster results than G4, inference can be accelerated on coprocessors (GPUs, FPGAs, etc), avenue to utilize HPCs
  - ***Cons:*** may need large training datasets and time, extrapolation may be unreliable

- Different approaches are possible
  - ***Replace (part of) FullSim:*** increase speed while preserving physics accuracy
  - ***Replace (part of) FastSim:*** decrease speed (slightly) while increasing accuracy
  - ***End-to-end:*** map generated-to-reconstructed information with no dedicated simulation step



The future

A fertile field where enthusiasm and ideas are plentiful and funding is scarce

Outlook

- There is a strong case for detector simulation R&D
  - Detector simulation uses a significant fraction of the HEP computing resources
  - Recently published R&D work points to opportunities for code simplification, modernization, optimization, and adaptation for use on accelerators
  - Software tools are > 25 years old and may not run on 2030's computer platforms
- My vision is for a diverse and flexible detector simulation program
  - Strong Geant4 development teams in the US
    - Alternative is to outsource detector simulation support of the domestic program
  - R&D program with heterogeneous computing and AI components
    - Flexibility to reallocate resources quickly based on lessons learned
    - A consensus on common solutions across institutional and international boundaries must follow the initial pilot and demonstrator phase
  - Any Geant4-in-GPU-based solution will require a multi-FTE/year effort for many years if the targets are HL-LHC and DUNE – continuous, sustainable effort

References

[0] V. Daniel Elvira, Impact of detector simulation in particle physics collider experiments, Physics Reports 695 (2017) 1-54. DOI: 10.1016/j.physrep.2017.06.002. https://arxiv.org/abs/1706.04293

[1] R. Ford, W. Nelson, The EGS Code System - Version 3, Stanford Linear Accelerator Center Report SLAC-210.

[2] R. Brun, F. Bruyant, M. Maire, A. McPherson, P. Zanarini, Geant3 CERN- DD-EE-84-1.

[3] A. Ferrari, P. Sala, A. Fasso, , J. Ranft, FLUKA: a multi-particle transport code, CERN-2005-10 (2005), INFN/TC 05/11, SLAC-R-773.

[4] The MARS Code System http://mars.fnal.gov.

[5] S. Agostineli et al. (Geant4 Collaboration), Gean4-a simulation toolkit, Nucl. Instrum. Meth. A 506 (2003) 250–303. doi:10.1016/S0168- 9002(03)01368-8.

[6] J. Allison et al. (Geant4 Collaboration), Recent developments in Geant4, Nucl. Instrum. Meth. A 835 (2016) 186–225. doi:10.1016/j.nima.2016.06.125.

[7] G. Amadio et al., GeantV: Results from the prototype of concurrent vector particle transport simulation in HEP, submitted to Software and Computing for Big Science, https://arxiv.org/abs/2005.00949v3

[8] Celeritas presentation in the Geant4 R&D Forum, April 14th 2020: https://indico.cern.ch/event/904385/

[9] AdeTP presentation in the Geant4 R&D Forum, September 15th 2020: https://indico.cern.ch/event/942142/sessions/363813/#20200915

[10] G4Opticks presentation in the Geant4 R&D Forum, September 15th 2020: https://indico.cern.ch/event/942142/sessions/363813/#20200915

[11] S. Blyth, Opticks : GPU Optical Photon Simulation for Particle Physics using NVIDIA® OptiXTM, EPJ Web of Conferences 214, 02027 (2019) https://doi.org/10.1051/epjconf/201921402027