Geant4 Version 10: Lessons learnt

A. Dotti for the Geant4 collaboration

Forum on Concurrent Programming Models and Frameworks

Wednesday, 15 January 2014

U.S. DEPARTMENT OF ENERGY
Office of Science

SLAC NATIONAL ACCELERATOR LABORATORY

Design Considerations: the process to G4 Version 10

- We first defined our **main design goals**:
  - Make efficient use of many-core technologies **reducing memory usage** w.r.t. MP, CoW, …
  - New Geant4 should be an **evolution** from current G4
  - User migration cost should be as **minimal** as possible
- A long (3 years) prototyping phase with well defined **incremental goals**:
  1. First define the technology
  2. Produce stand-alone code (e.g. branched from G4) showing main functionalities
  3. Demonstrate scalability of solutions
  4. Integrate in main code-base
  5. Release and patch

Design Considerations: the process to G4 Version 10

- MT code integrated into G4

- Public release
- All functionalities ported to MT

- Proof of principle
- Identify objects to be shared
- First testing

- API re-design
- Example migration
- Further testing
- First optimizations

- Further Refinements
- Focus on further performance improvements

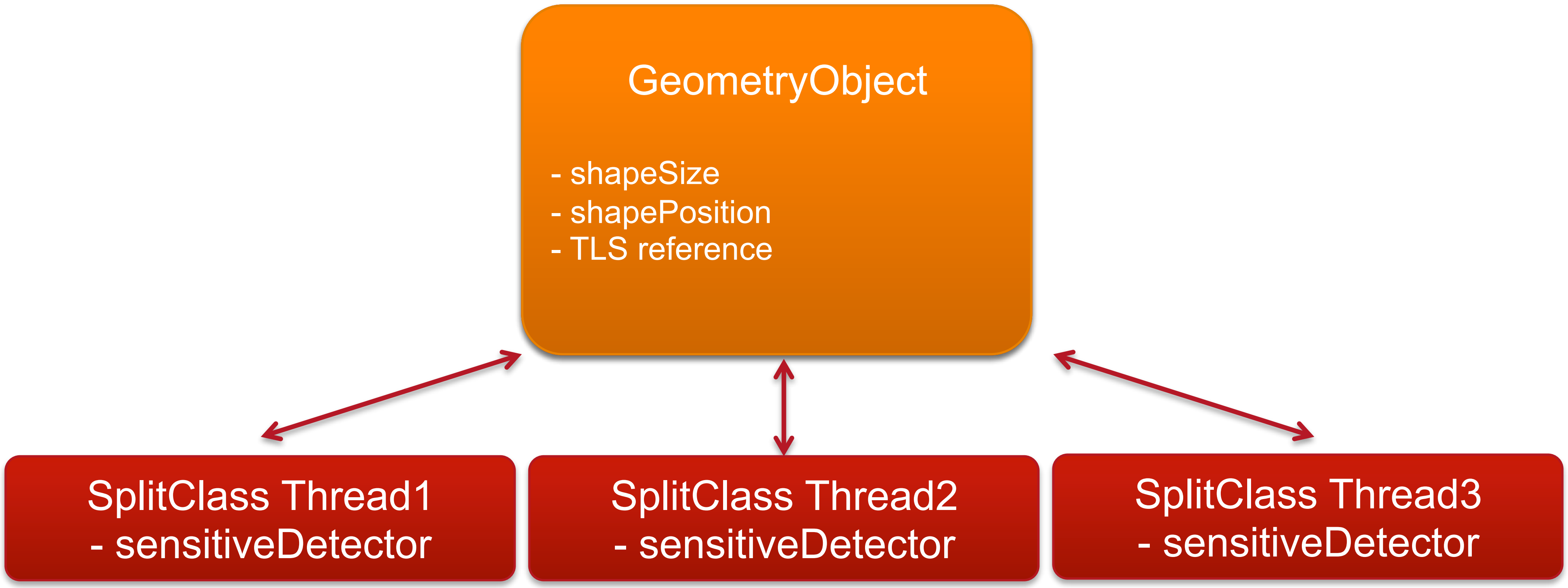
Where to start from?

- In MC simulation events are independent, natural choice: event-level parallelism
  - **Requires all code to be thread-safe**, two options possible:
    1. Review each single class in G4
    2. Develop a general strategy that is valid everywhere
- Second option requires **long initial design/prototyping work** but it is more beneficial in the long run (thread-local-storage)
- After this general design phase we focused on reducing memory footprint (split-classes)

Thread-safety in Version 10.0

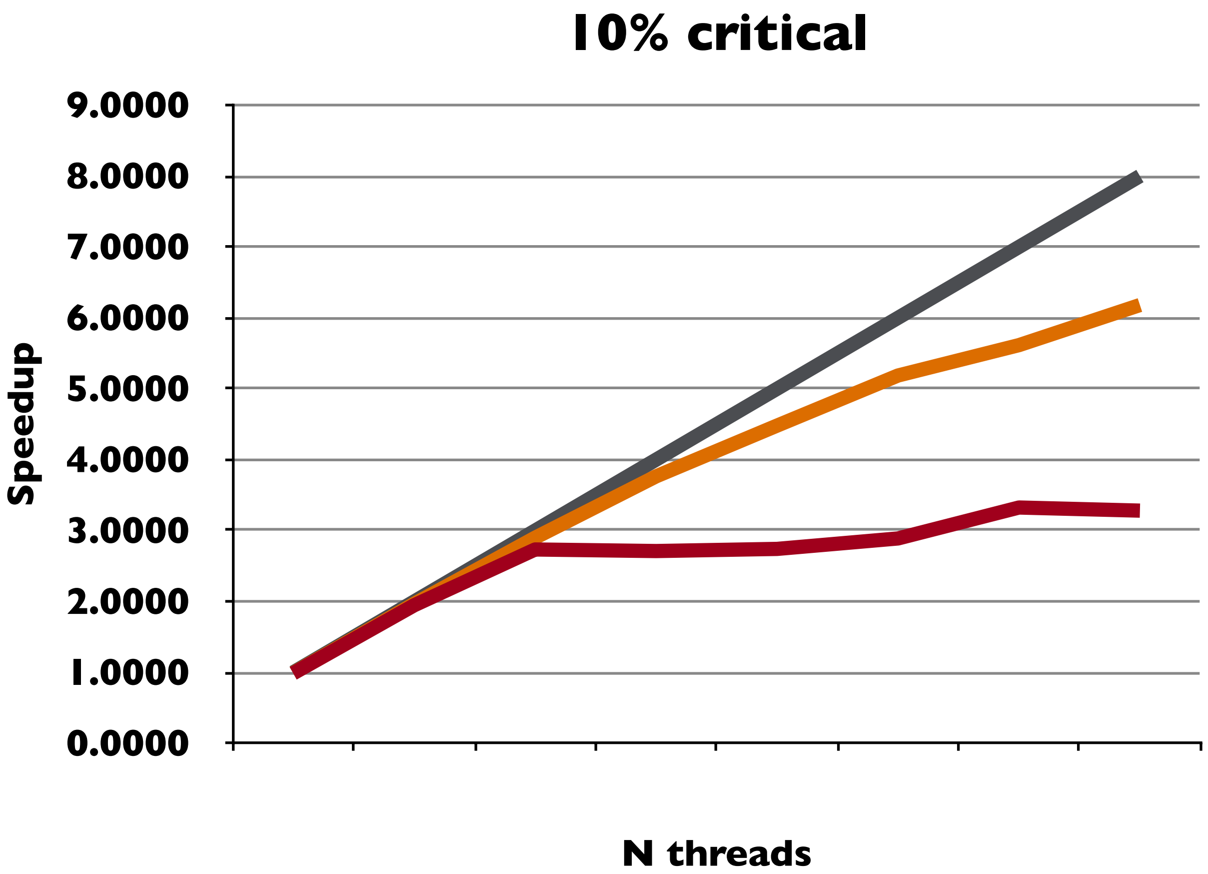
SLAC

- Thread-safety implemented via **Thread Local Storage**
  - Managers (e.g. singleton) are basic components: "naturally" thread-local
- **"Split-class" mechanism:** reduce memory consumption
  - Read-only part of most memory consuming objects shared between thread: Geometry, Physics Tables
  - Rest is thread-private



Thread Local Storage

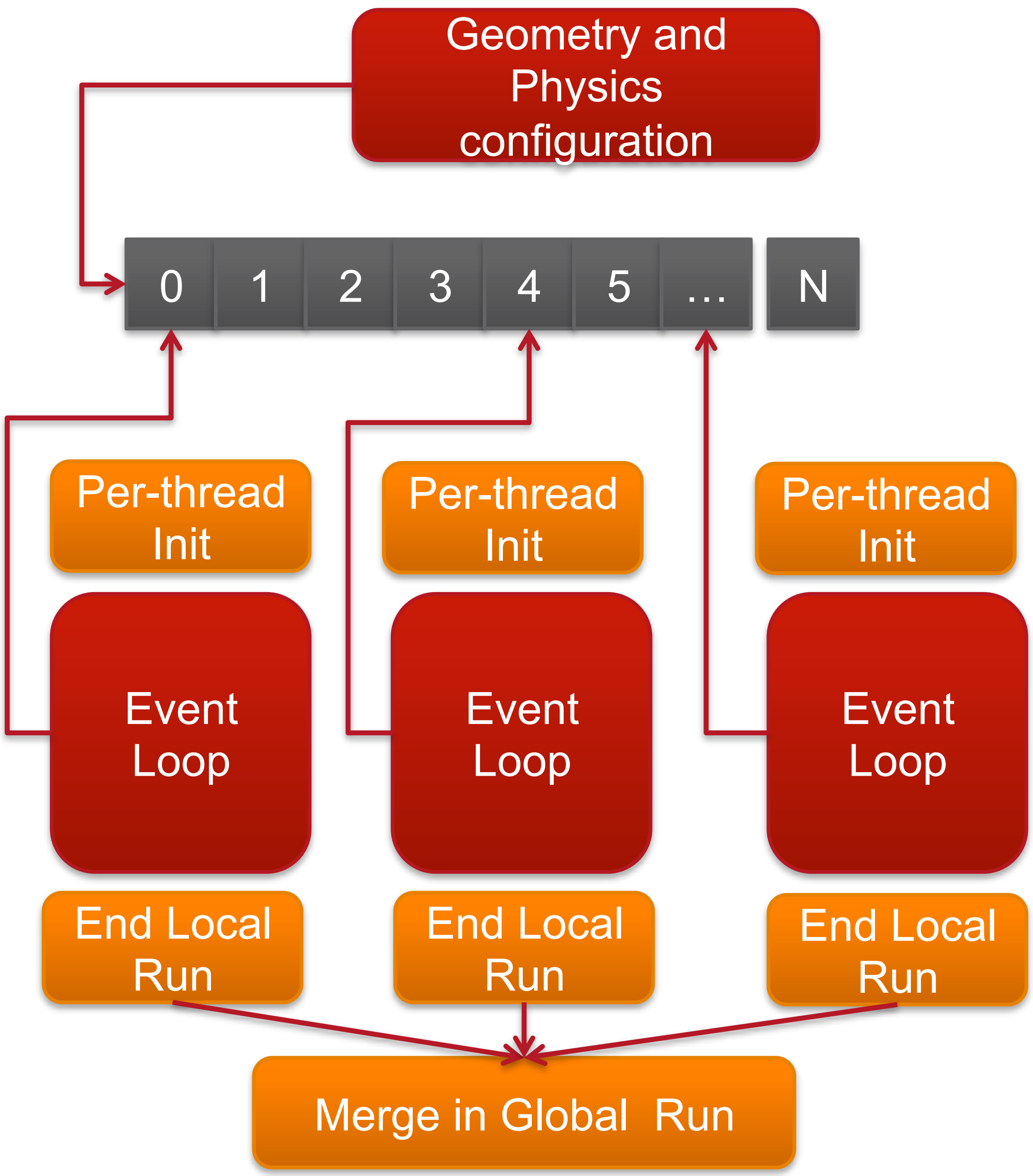
SLAC



- Each (parallel) program has sequential components
  - **Protect access to concurrent resources**
- Simplest solution: use mutex/lock
- TLS: each thread has its own object (no need to lock)
  - **Supported by all modern compilers**
- Challenge: only simple data types for static/global variables can be made TLS
- Warning: hidden locks are important too (e.g. `operator new`, use of `std::strstream`)

NB: results obtained on toy application, not real G4

Multi-threading master/worker model



Per-event seeds pre-prepared in a "queue"

Threads compete for next event to be processes (new in ref-08)

Command line scoring and G4tools automatically merge results from threads

Did we get it right?

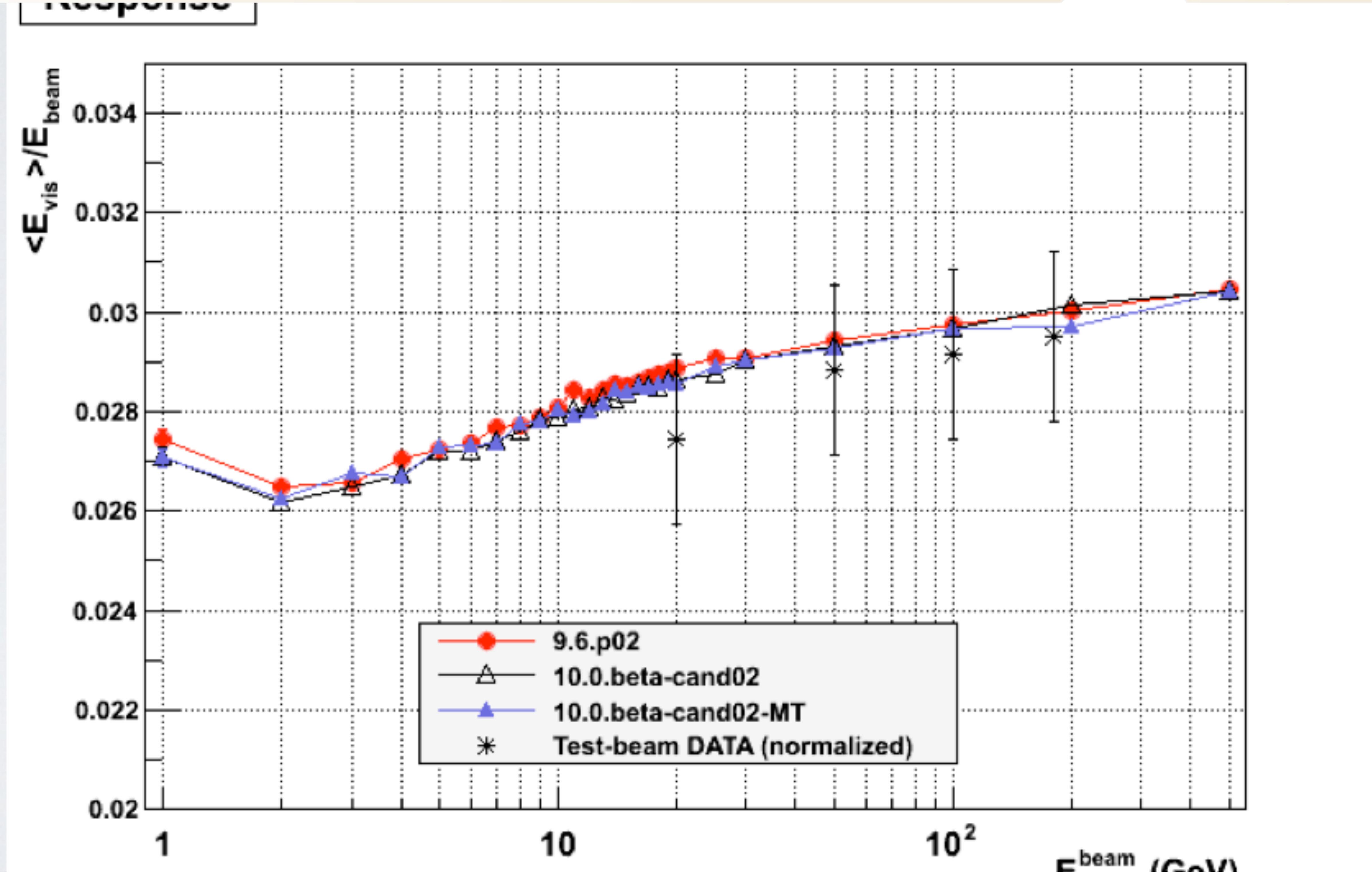
- Constantly review design and implementation choices with partners:
  - **Developer community:** dedicated mailing-list for discussions, twiki pages, in-depth face-to-face discussions
  - **User community**: release soon and often, setup a dedicated user-community forum (31 threads, 27 are related to prototypes)
  - **"Official" documentation**: conference proceeding, articles, manuals (extremely important also for developers)
- We are not expert in the sector: **collaboration with Computing Scientist**
  - Collaboration between physicists (authors of algorithms) and computing scientists (experts in how to efficiently implement them) is a key element of G4-MT success
  - It also helps in reducing typical physicists attitude to "reinvent the wheel"

Focus on important metrics

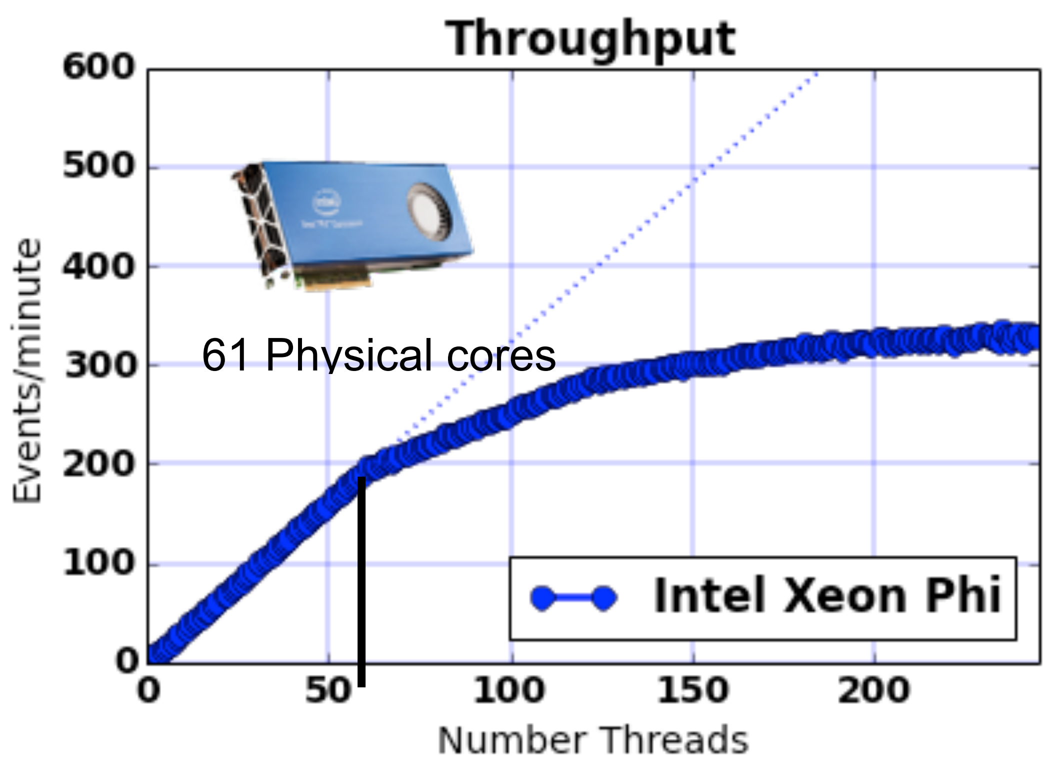
SLAC

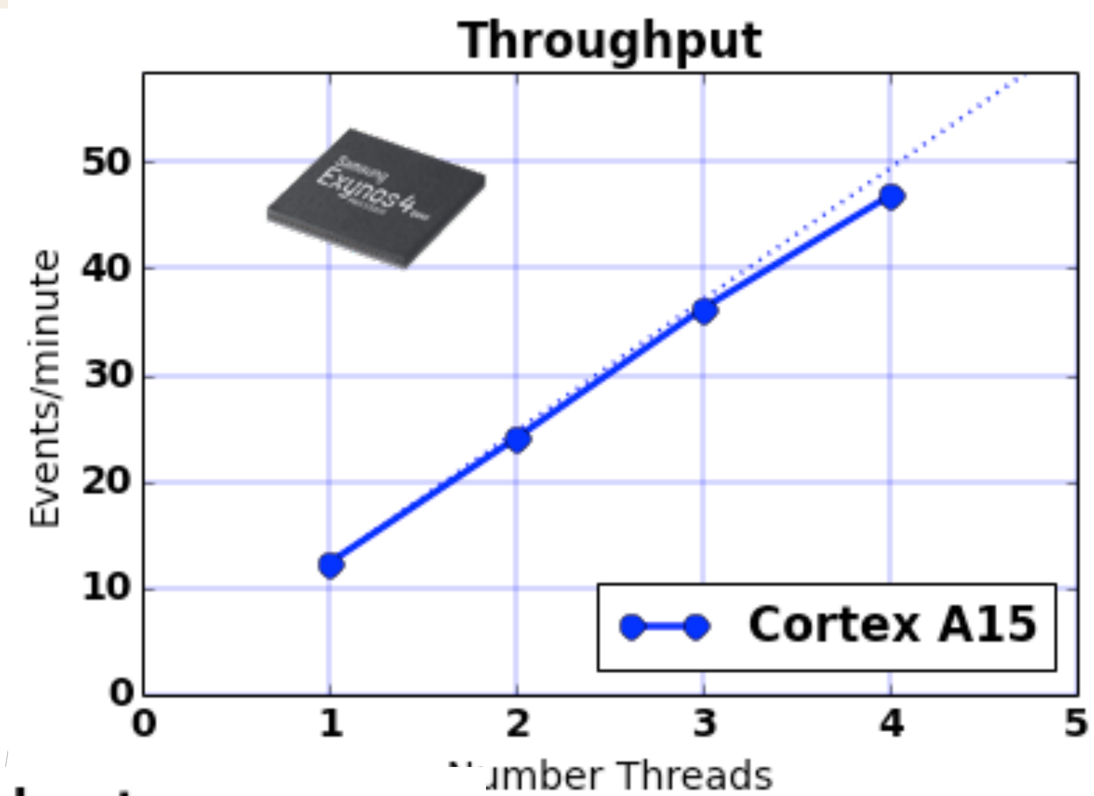
- Defined **very early** our goals and how to measure them:
  - Produce **physics results** equivalent to sequential code independently of number of threads and event simulation order: Strong Reproducibility (much more difficult than what it sounds)
  - Two main metrics: **linearity of speedup**; **memory reduction**
  - Define immediately few test-benches (SimplifiedCalorimeter and FullCMS), independent group responsible for monitoring
- We have learned a lot from **using very early different hw and sw systems**: x86_64, MIC, ARM, Atom architectures
  - Linux, Mac OS X
  - Initial plan included also WIN, on-halt due to man power, challenging due to non POSIX standards
- **Measure often:** at least once per month

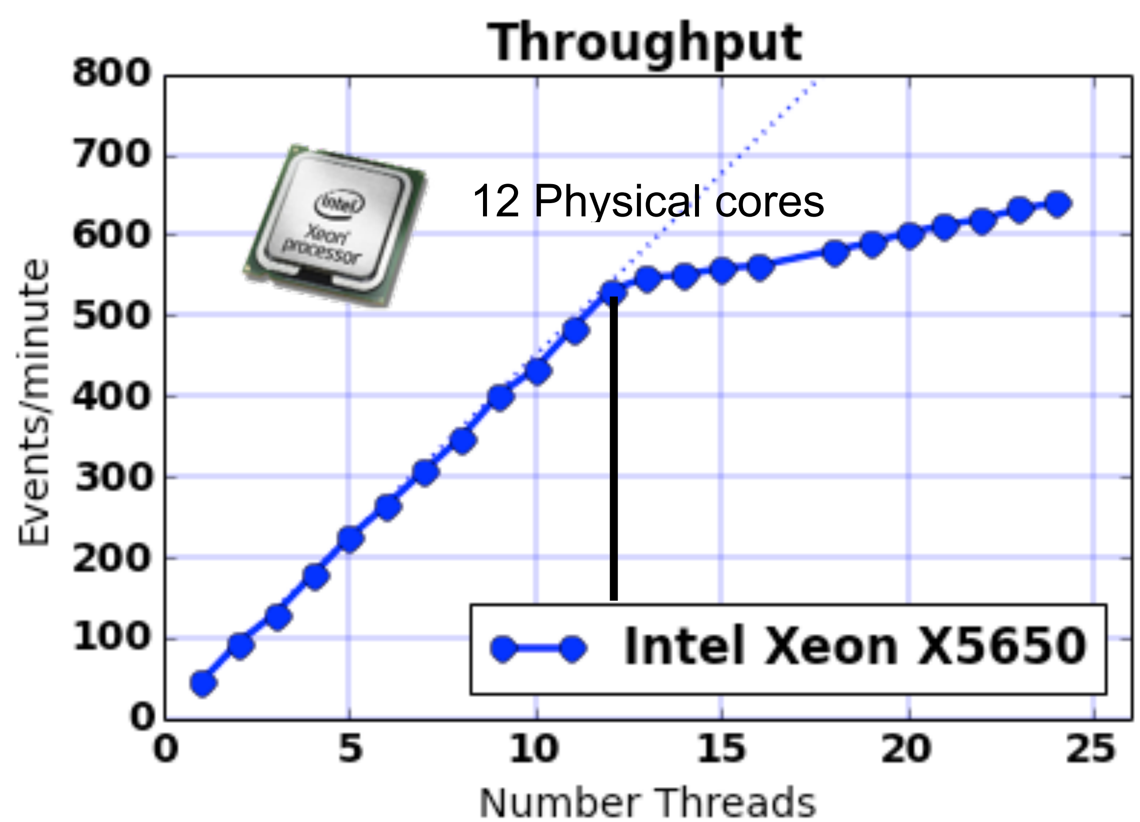
MT libs Vs SEQ libs



Performances





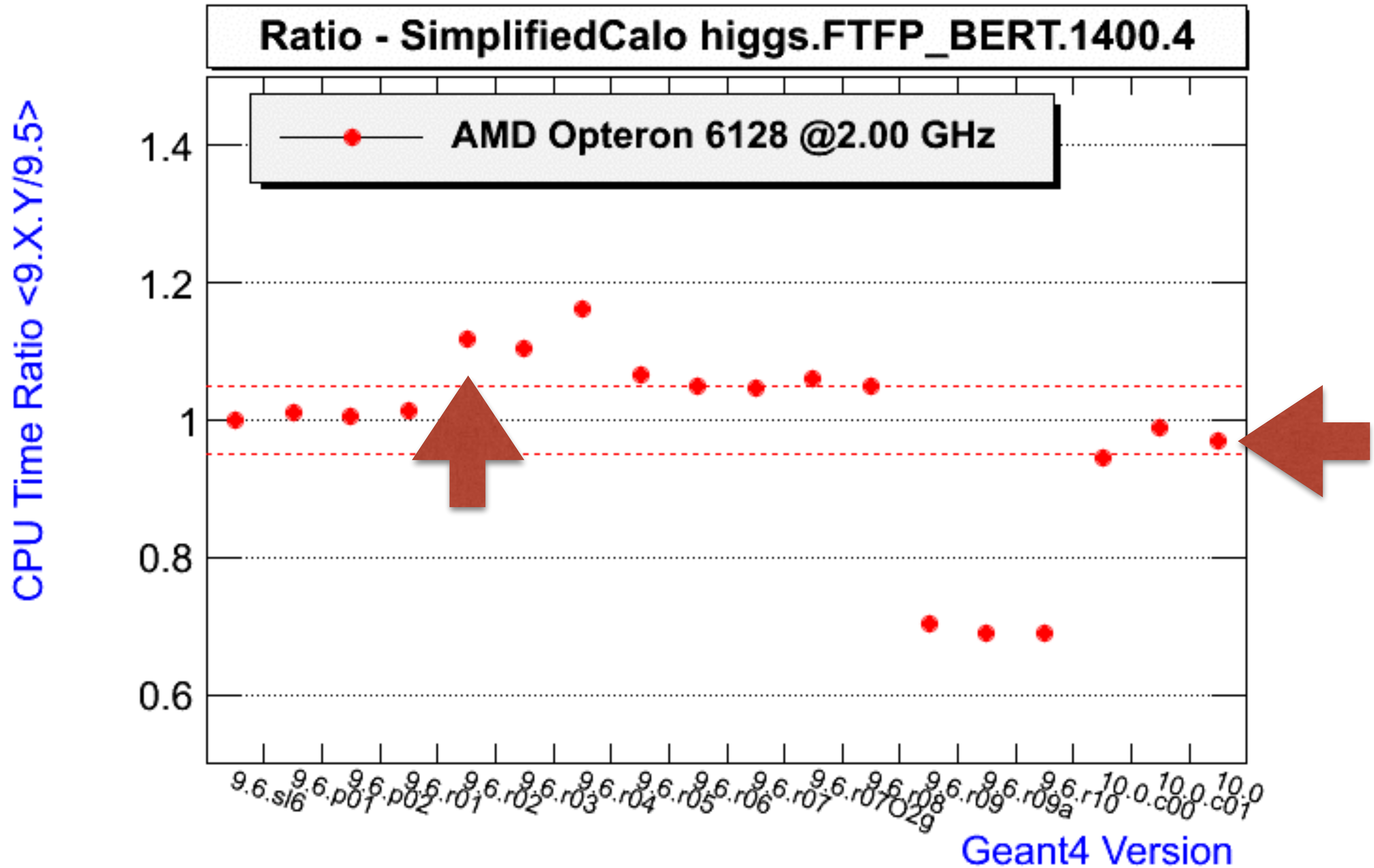


Absolute performances

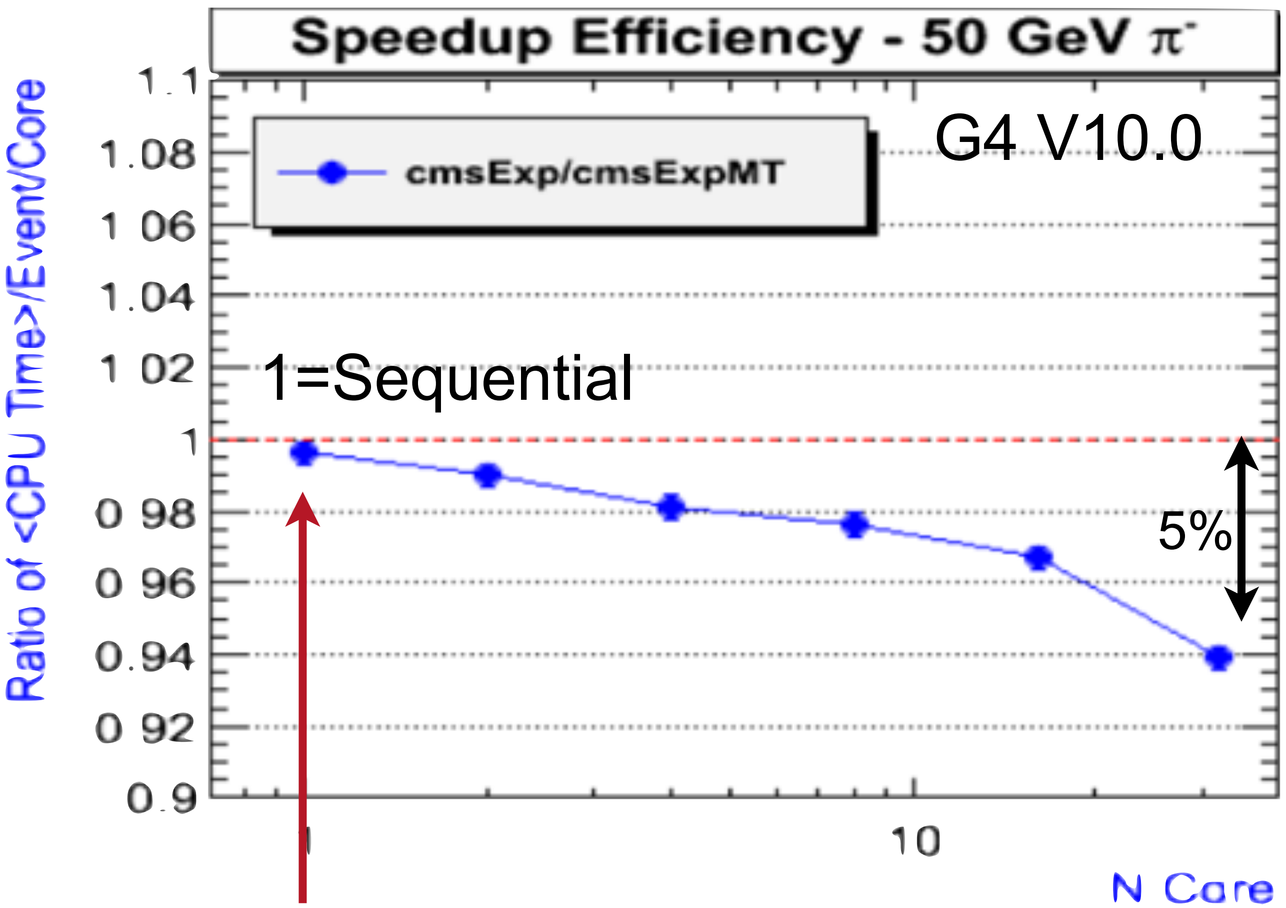
- We did expect a penalty for MT (e.g. performances 1-thread < sequential): due to TLS "machinery"
  - Challenge: **understand TLS details**, relatively new feature and documentation not always complete/clear.
- All initial performance issues **have been solved:**
  - There is only very little CPU penalty for MT builds
  - TLS is very powerful when used correctly, but should not be over-used

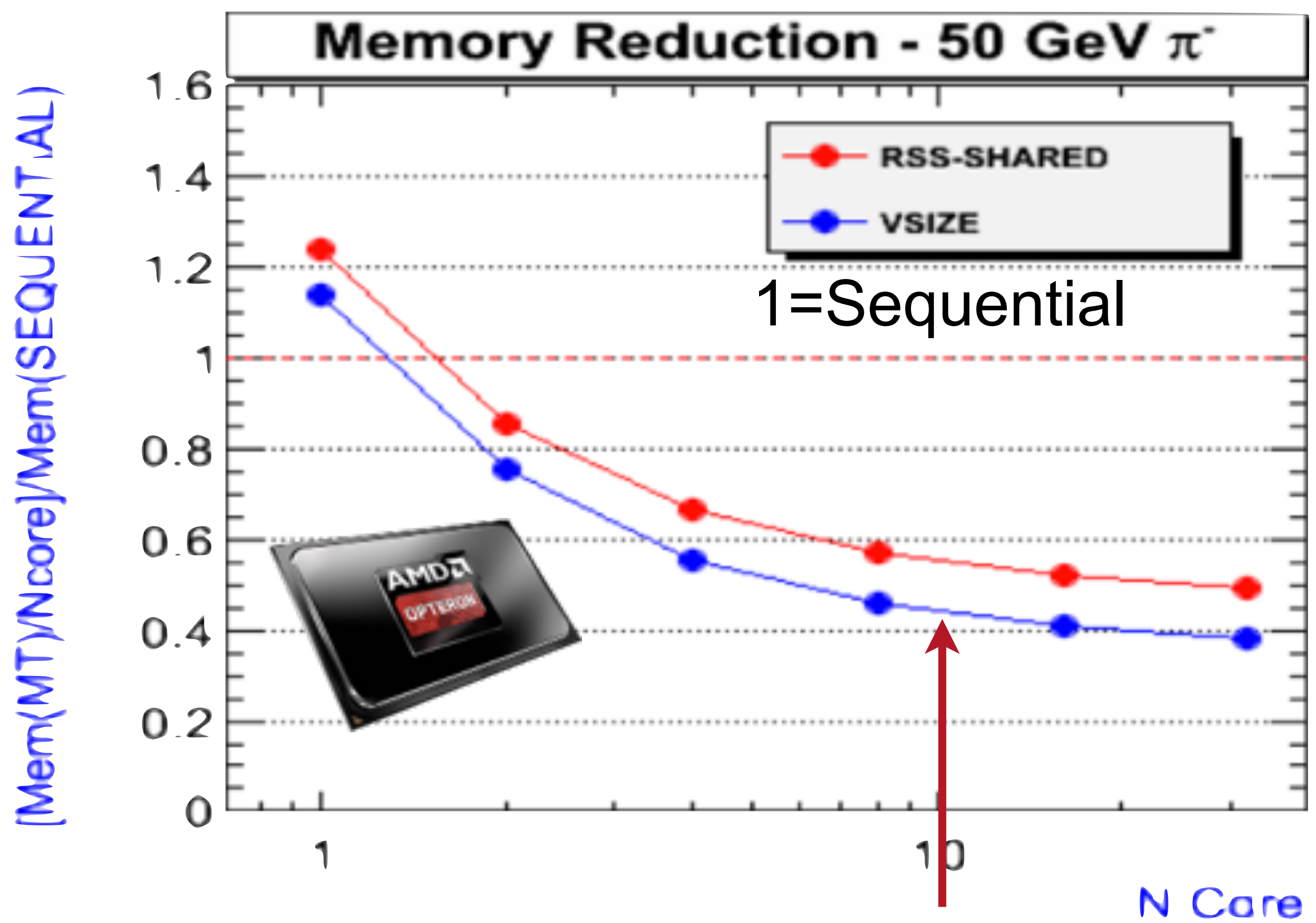
- Improvements in MT often brought benefits **also to sequential applications** (V10.0 w/ improved physics, MT functionalities, is faster than previous releases):
  - Re-arranging memory layout of geometry and physics (split-classes) bring some benefits in some cases (improve cache hit ratio)
  - Forced us to review several areas of our code
  - Would have more difficult with a separate "code-base" or using ad-hoc technologies/ languages

Benefits of MT developments for sequential code



Comparing with sequential





1 thread =>
Overhead for MT
Very small CPU penalty
~1%

10 threads =>
50% memory w.r.t.
10 sequential instances

Tools

- Finding right tools to develop MT code is a **challenge:**
  - **Development**: IDE with integrated GDB sessions are useful (but not full support for TLS), use coverity to fix possible defects, extensive use of CDash/CTest
  - **Debug**: we could not find any tool specifically designed to help debug MT applications
    - Most challenging aspect: crashes are non-reproducible, stack-traces are not always useful
    - Need a lot of experience: often developers ask the "experts" to re-run and debug
    - DRD is very useful for data-race identification (some experience needed to interpret output)
  - **Performances/profiling:** simple benchmarking is very useful if done often, full profiling tools need to be MT aware (OpenSpeedShop, Gooda)
  - Interesting lesson: typical "sequential" tricks to speed up simulations (e.g. caching calculations), may be not be so beneficial with many threads (increase hw cache misses). How to define tradeoff?
- G4 comes with extensive set of tests/examples (240 are run nightly in cdash), earlier MT migration would have helped to spot issues/bug earlier (lesson learnt: some sort of **test-driven development** would have been beneficial)

Gooda Example

SLAC

Reports

- reports/Sample
- reports/G4MT-new
- reports/G4MT-old

reports/G4MT-new Hotspots

Cycles Samples

Enter search term

process path	module path	unhalted_core_cycles		uops_retired:stall_cycles		instruction_retired	uops_retired:any
		8245843	(100%)	5703358	(69%)	3905966	4766639
ParFullCMS		8229714	(99%)	5654865	(68%)	3900553	4757241
aggregated_kernel_object		91525	(1%)	111330	(121%)	17110	28074
vmlinux		4869	(0%)	40659	(835%)	1492	4067
puppet		1431	(0%)	855	(59%)	556	723
perf		1401	(0%)	979	(69%)	563	881
sshd		934	(0%)	460	(49%)	873	1107
kworker/9:0		358	(0%)	248	(69%)	87	90
khugepaged		308	(0%)	190	(61%)	71	113
kworker/7:0		308	(0%)	234	(75%)	56	90
kworker/0:0		298	(0%)	190	(63%)	48	83
kworker/14:2		278	(0%)	285	(102%)	56	105
kthreadd		268	(0%)	234	(87%)	40	113

Cycles Samples

Enter search term

function name	unhalted_core_cycles		uops_retired:stall_cycles		instruction_retired	uops_retired:any	load_latency	
	8245843	(100%)	5703358	(69%)	3905966	4766639	2745332	(33%)
G4PhysicsVector::Value(do…	249801	(3%)	183440	(73%)	81361	96333	157004	(62%)
G4ElasticHadrNucleusHE::H…	175220	(2%)	110935	(63%)	119668	177152	2405	(1%)
G4Navigator::LocateGlobal…	190355	(2%)	141634	(74%)	59378	74117	144841	(76%)
G4PolyconeSide::DistanceA…	134038	(1%)	80485	(60%)	99852	125190	14410	(10%)
G4SteppingManager::Define…	131792	(1%)	89575	(67%)	79845	87085	75913	(57%)
G4CrossSectionDataStore::…	121696	(1%)	78271	(64%)	73528	96325	73975	(60%)
CLHEP::MTwistEngine::flat…	107107	(1%)	60097	(56%)	89551	93810	30359	(28%)
G4ClassicalRK4::DumbStepp…	82552	(1%)	37933	(45%)	113510	131667	2723	(3%)
G4Navigator::ComputeStep(…	110755	(1%)	76064	(68%)	52331	68123	28749	(25%)
G4VoxelNavigation::Comput…	113825	(1%)	80083	(70%)	50370	58626	62875	(55%)

function name	instruction_starvation		bandwidth_saturated		branch_misprediction		store_resources_saturated		instruction_latency		exception_handling	
	4665713	(56%)	80157	(0%)	522052	(6%)	204496	(2%)	1724700	(20%)	41728	(0%)
G4PhysicsVector::Value(do…	102099	(40%)	3478	(1%)	16477	(6%)			72405	(28%)	606	(0%)
G4ElasticHadrNucleusHE::H…	3021	(1%)	70	(0%)	884	(0%)	467	(0%)	123127	(70%)	1660	(0%)
G4Navigator::LocateGlobal…	101920	(53%)	1958	(1%)	9192	(4%)	7890	(4%)	25192	(13%)	586	(0%)
G4PolyconeSide::DistanceA…	10643	(7%)	1153	(0%)	6291	(4%)	20	(0%)	58860	(43%)	636	(0%)
G4SteppingManager::Define…	68719	(52%)	358	(0%)	7225	(5%)	3597	(2%)	15542	(11%)	407	(0%)
G4CrossSectionDataStore::…	36391	(29%)	258	(0%)	4541	(3%)	21048	(17%)	13356	(10%)	318	(0%)
CLHEP::MTwistEngine::flat…	41231	(38%)	179	(0%)	7622	(7%)			15185	(14%)	447	(0%)
G4ClassicalRK4::DumbStepp…	9948	(12%)	30	(0%)	2335	(2%)	6141	(7%)	17739	(21%)	199	(0%)
G4Navigator::ComputeStep(…	103898	(93%)	626	(0%)	9063	(8%)	298	(0%)	11041	(9%)	417	(0%)
G4VoxelNavigation::Comput…	55998	(49%)	745	(0%)	8238	(7%)	4074	(3%)	11538	(10%)	398	(0%)

Technology Choices

SLAC

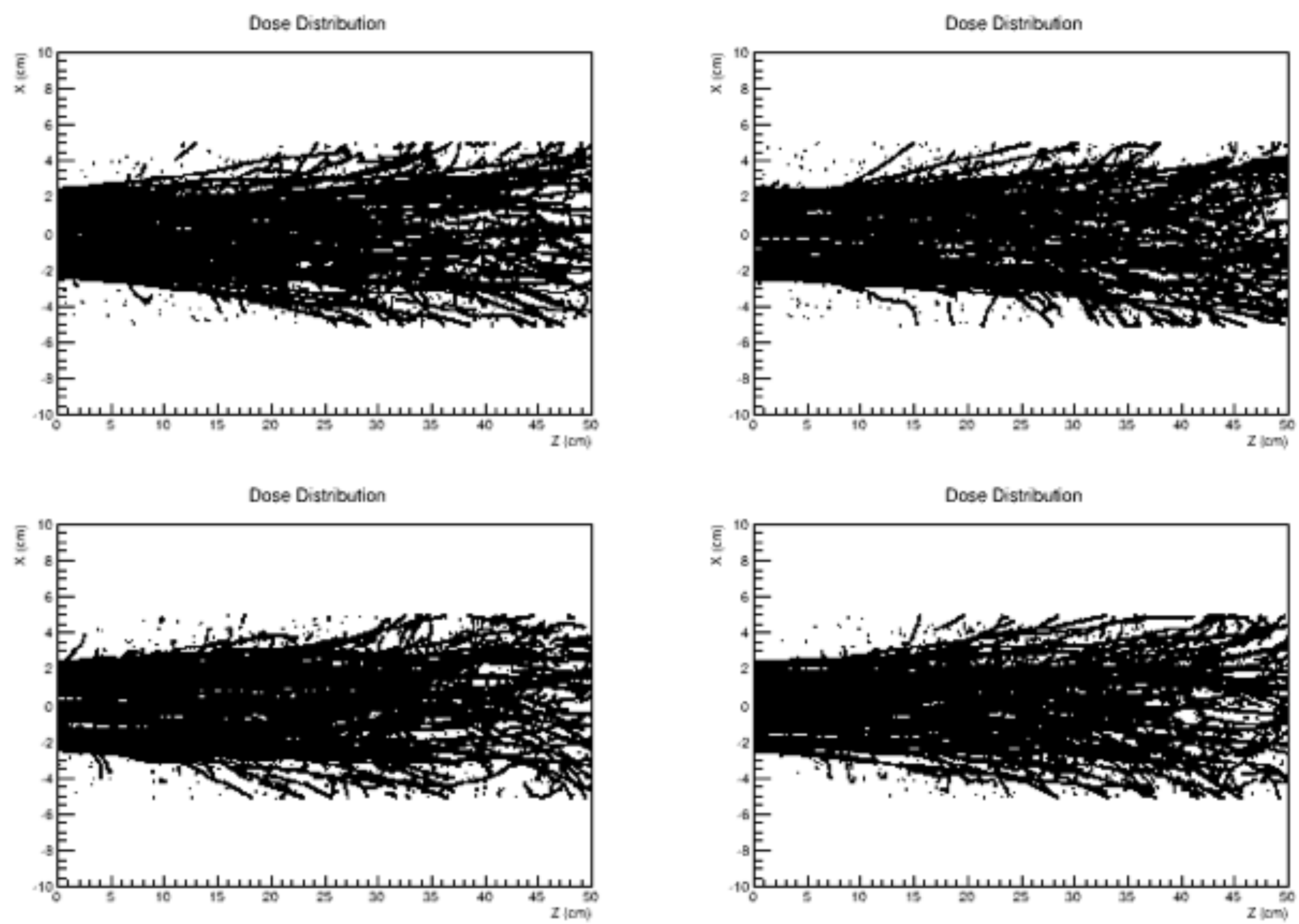
- We did **quite conservative** (sw) technology choices:
  - Use only POSIX standard (e.g. pthreads)
  - Use only compiler supported features (e.g. TLS)
- We believe these are **good choices:**
  - Almost trivial to port to new architecture (e.g. MIC porting done in few days)
  - Allow for **integration with "frameworks"** (provided that are compatible w/ standards): we have examples integrating MPI and TBB
  - Remember Geant4 is a toolkit integrated in larger (experimental) frameworks, not the other way around
- Very important: since these are standards, lots of example and documentation, lots of experience from other fields

Heterogeneous parallelism: MPI based G4MT

- **MPI based parallelism** available in Geant4
  - MPI works together with MT

Example:
4 MPI jobs
2 threads/job
MPI job owns histogram

Next Step:
Host + MIC simulation
Based on MPI



What's next?

SLAC

- **Further reduce memory consumption.** Thumb-rule: fit complex simulations on accelerators w/ O(100) threads and O(GB) memory
  - Warning: minimize memory usage can sometime **conflict** with other performance considerations (e.g. reduce memory "churn" may not always be thread-safety)
  - In our experience **profiling guided optimizations** are very effective: run profiling tools, identify top offender, work on them, repeat
- Further CPU benefits will come looking at single algorithms for new parallelization opportunities

Kind of Conclusions: My vision for Geant4 Version 11.0 :-)