

Geant4 Version 10

Status and outlook

A. Dotti for the Geant4 collaboration
HEP/ASCR Meeting; 5th February 2014

Status: Geant4 Version 10 (6 December 2013)



SLAC National Accelerator Laboratory

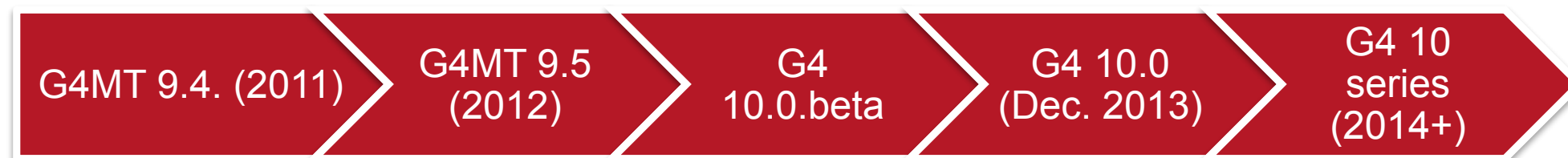
Photo Credit: Peter Ginter

Introduction

- **Event level parallelism** via multi-threading (POSIX based)
- Built on top of experience of G4MT prototypes
- Main design driving goal: **minimize user-code changes**
- Integrated into Version 10.0 codebase

- MT code integrated into G4

- Public release
- All functionalities ported to MT

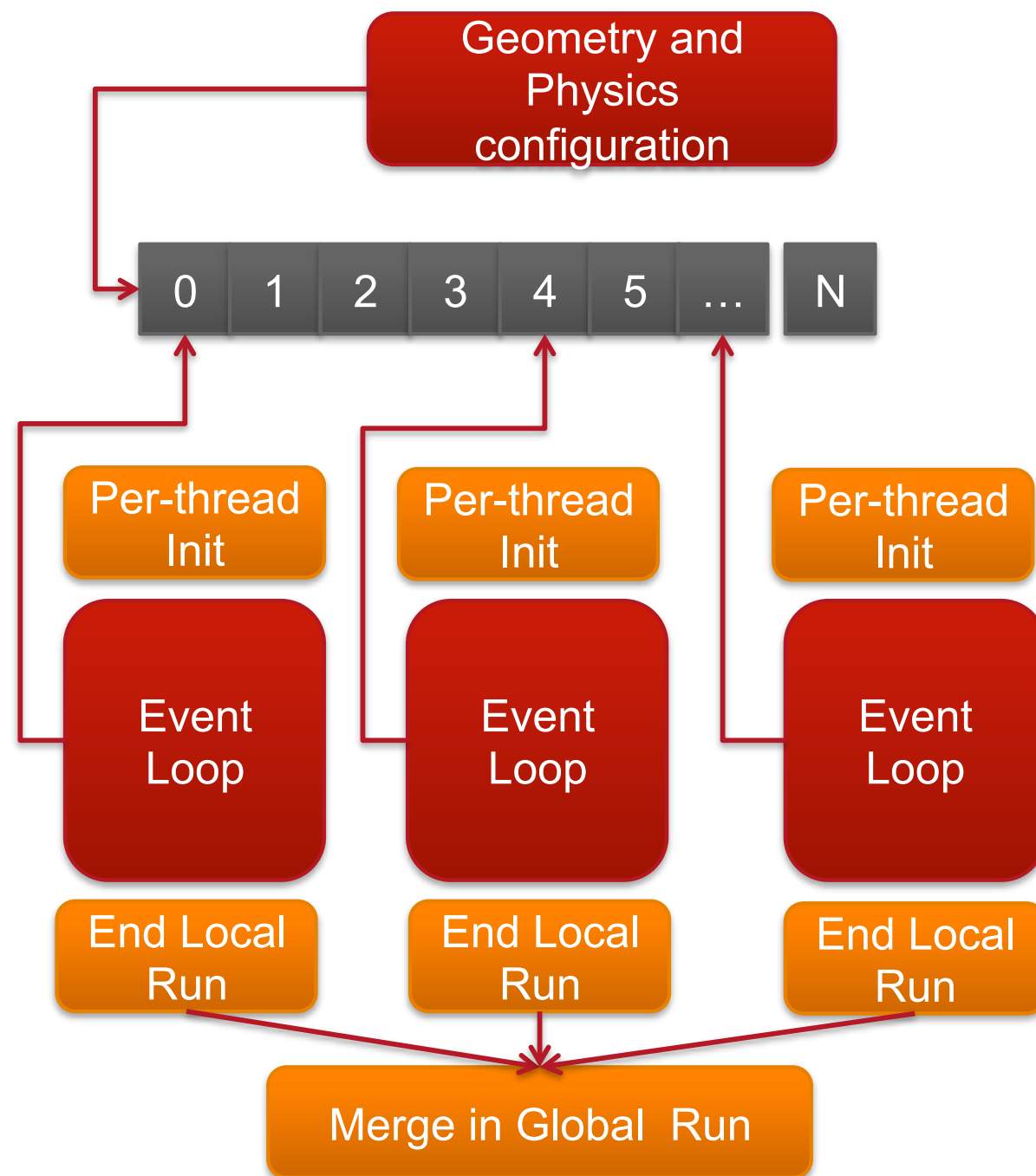


- Proof of principle
- Identify objects to be shared
- First testing

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

- Further Refinements
- Focus on further performance improvements

Multi-threading master/worker model



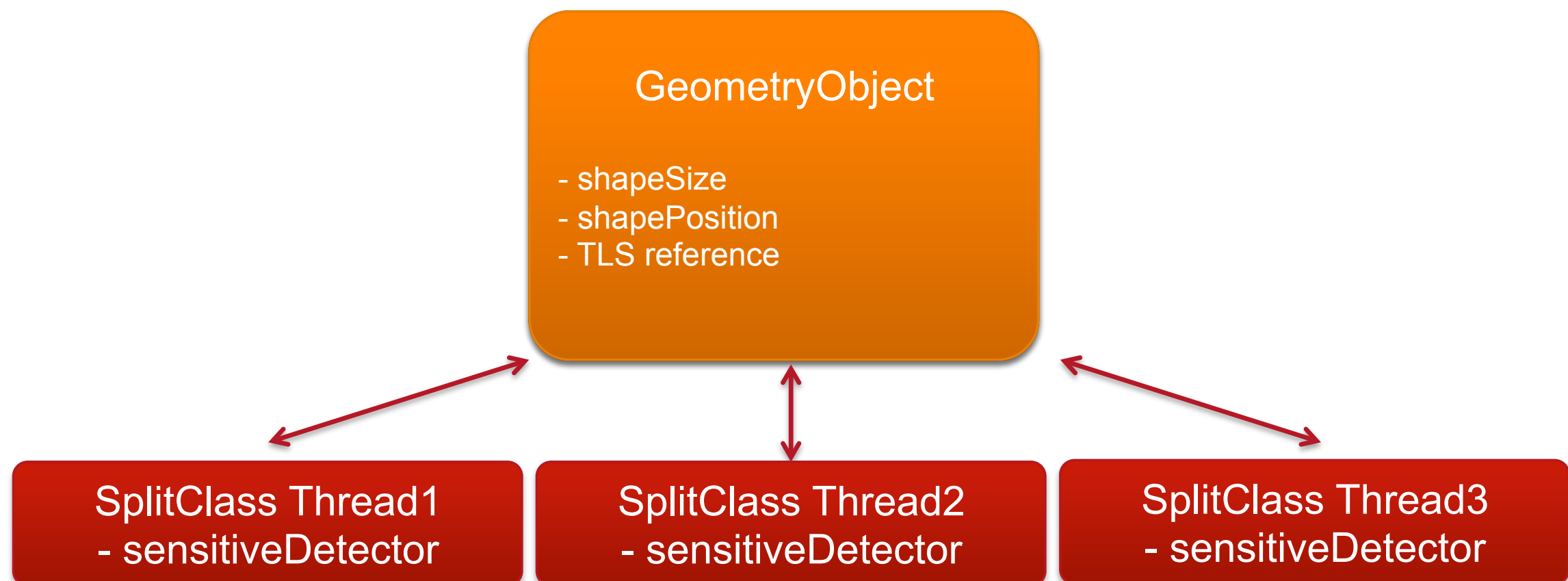
Per-event RNS seeds
pre-prepared:
guarantees
reproducibility

Threads compete for next
“bunch” of events. Optimal
bunch size is a parameter to
minimize locking

Command line scoring and G4
histo tools automatically perform
reductions at the end of the job.

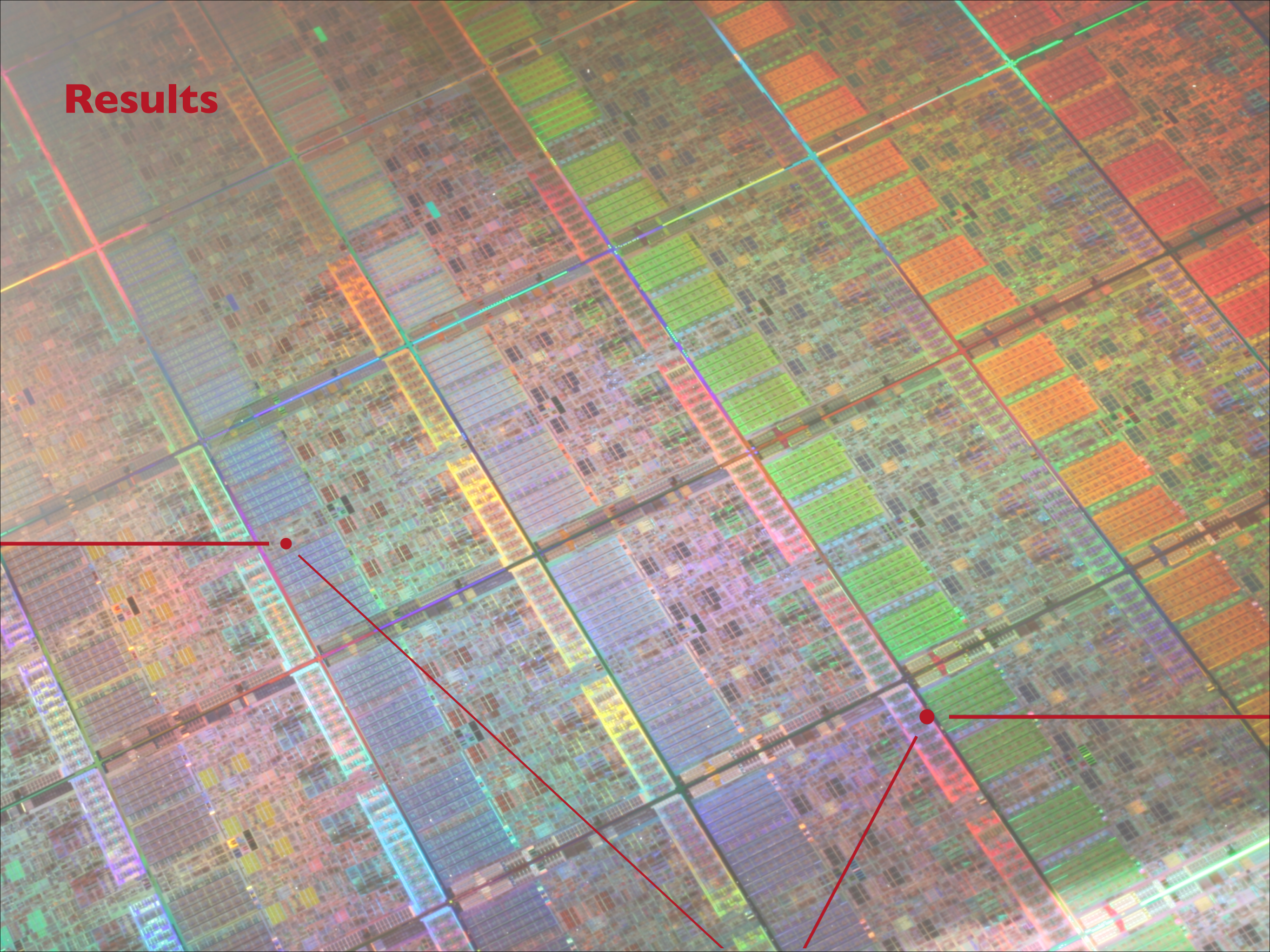
Thread-safety in Version 10.0

- Design: lock-free code during event-loop
- Thread-safety implemented via **Thread Local Storage**
- “Split-class” mechanism: reduce memory consumption
 - Read-only part of most memory consuming objects shared between thread: geometry, (EM) physics tables
 - Rest is thread-private





- **Very strong interest** from user community: 30 threads in new dedicated user-forum (not only HEP)
- CMS: interest in integration with **TBB-based experimental framework**. First simple TBB-based example provided (examples/extended/parallel/TBB)
- ATLAS: interest in evaluation MT in ISF (Integrated Simulation Framework) **mixing different flavors of simulation** (e.g. fast and full) and possibly in parallel
- ALICE: strong interest running in MT already during this year
- Uni Manchester: use of G4 on Xeon Phi for imaging and treatment planning

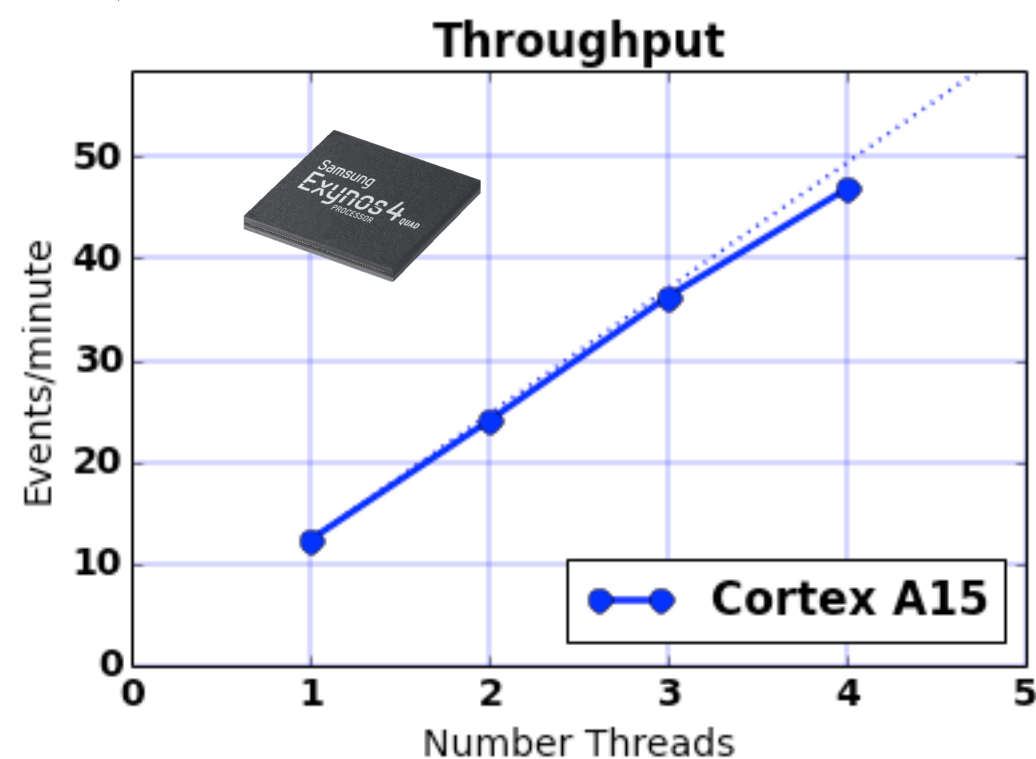
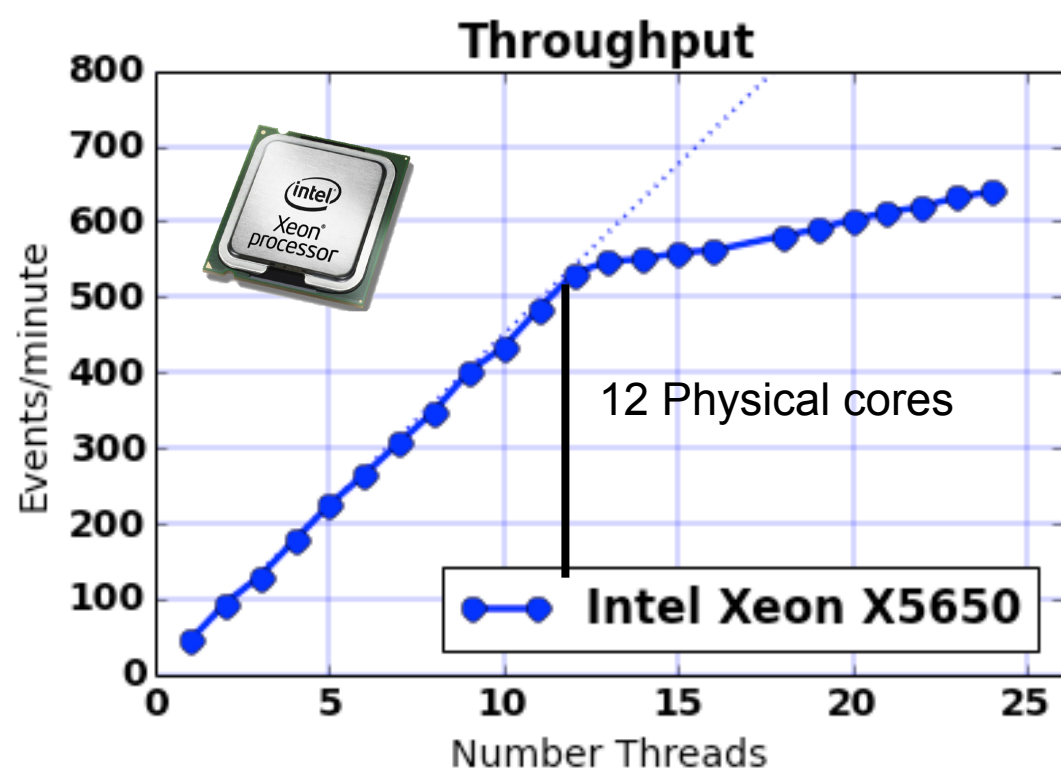
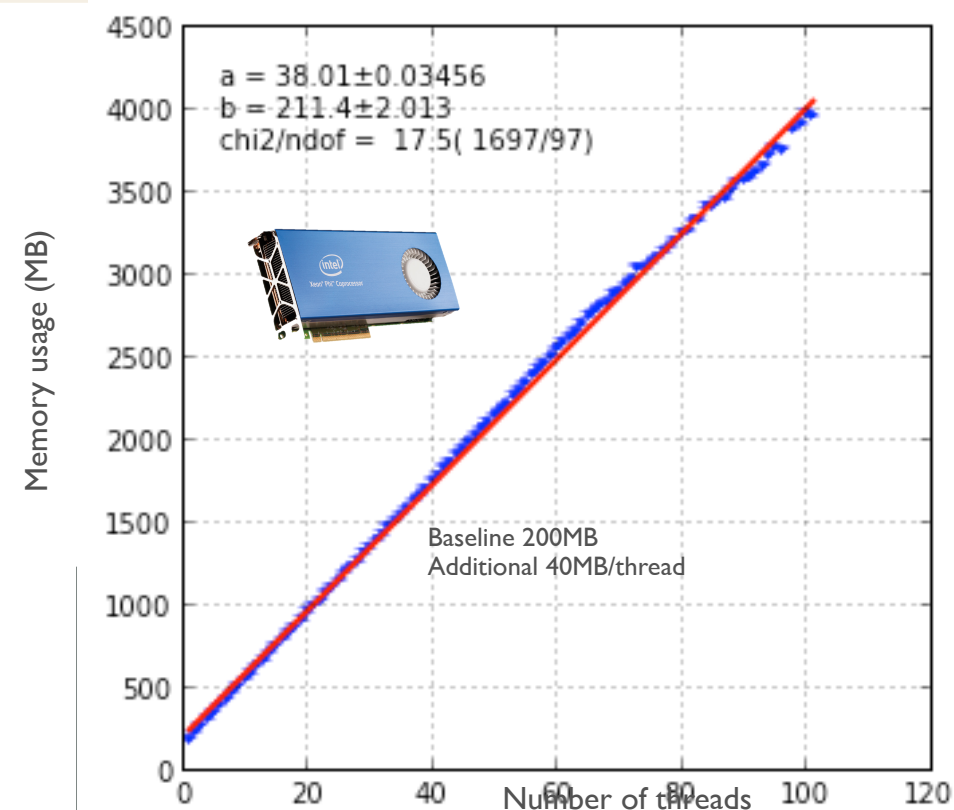
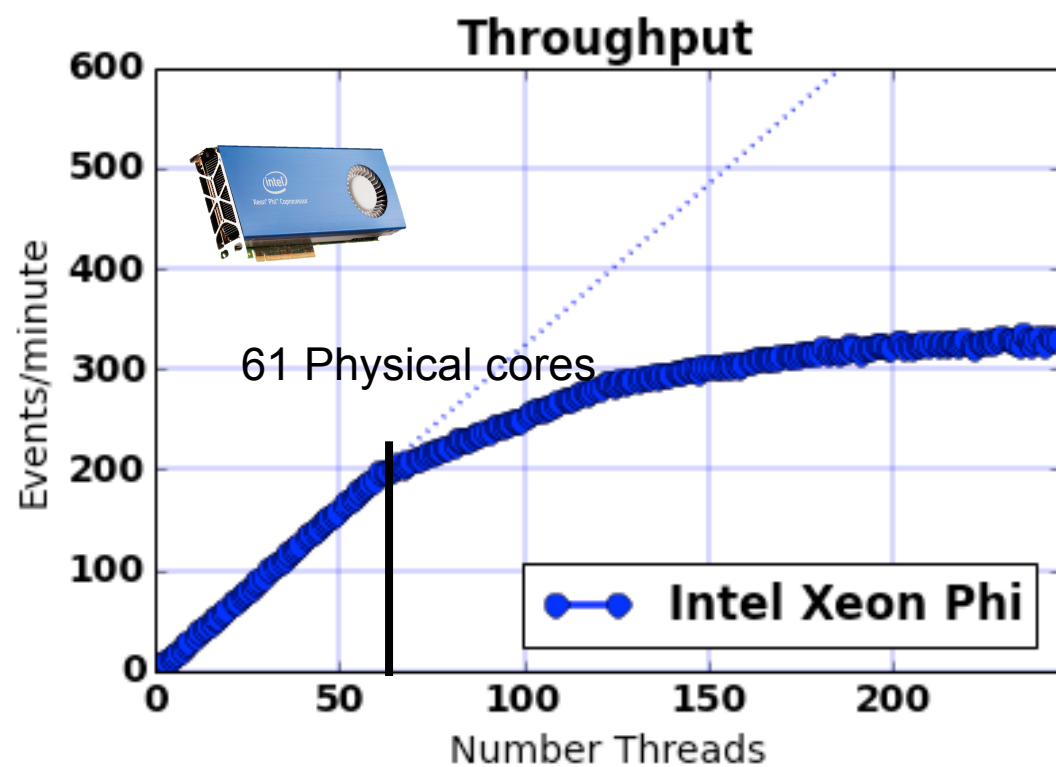
Results



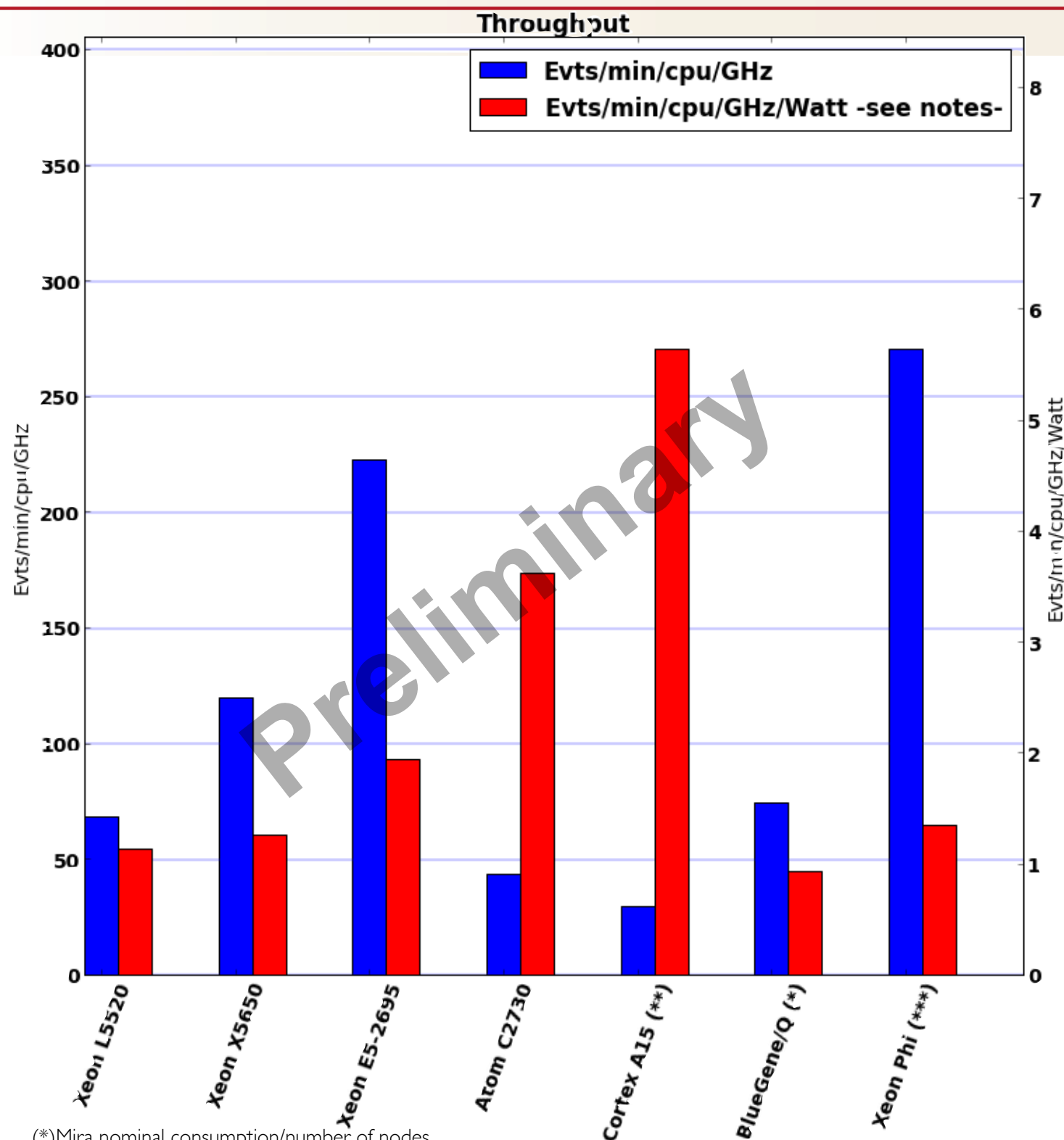
General considerations

- **Fully reproducible:** given an event and its initial seed the RNG history is independent of the number of threads and order in which these are simulated
 - Corollary 1: given the seeds, sequential and MT builds are equivalent
 - Corollary 2: being able to reproduce a single event in a dedicated job (i.e. crashes)
- MT functionality introduces **minimal overhead** ($\sim 1\%$) w.r.t. sequential
- Very good **linear speedup** up to very large number of threads $O(100)$
- Good **memory reduction**: only 30-50MB/thread (depends on application)
- Hyper-threading adds additional +20% throughput
- Working out-of-the-box with success on **different architectures** x86, ARM, MIC,  Atom,  IBM Bluegene/Q

Results



Cross-comparing architectures



- Throughput normalized per GHz and “socket” (or node / card)
- Not a measure of the absolute performance of a system
- Also reported Throughput/Watt: not realistic (mainly not counting server, very rough!) only to give an idea of what we are talking about
- What is the best “metrics” to compare different architectures?

Absolute performances:

```

===== Max Events/min/cpu =====
154.4619 Intel Xeon L5520@2.27GHz
319.7392 Intel Xeon X5650@2.67GHz
534.6305 Intel Xeon E5-2695 v2@2.40GHz
73.8040 Intel Atom C2730@1.7GHz
46.8705 Exynos 5410 Octa Cortex-A15@1.6GHz
119.2088 BlueGene/Q@1.6GHz
334.4548 Intel Xeon Phi 7120P@1.238GHz
    
```

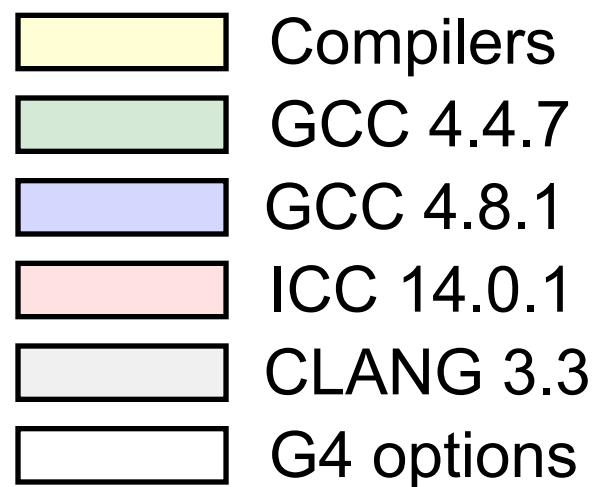
(*)Mira nominal consumption/number of nodes

(**) Measured for a ODROID-XU+E evaluation board

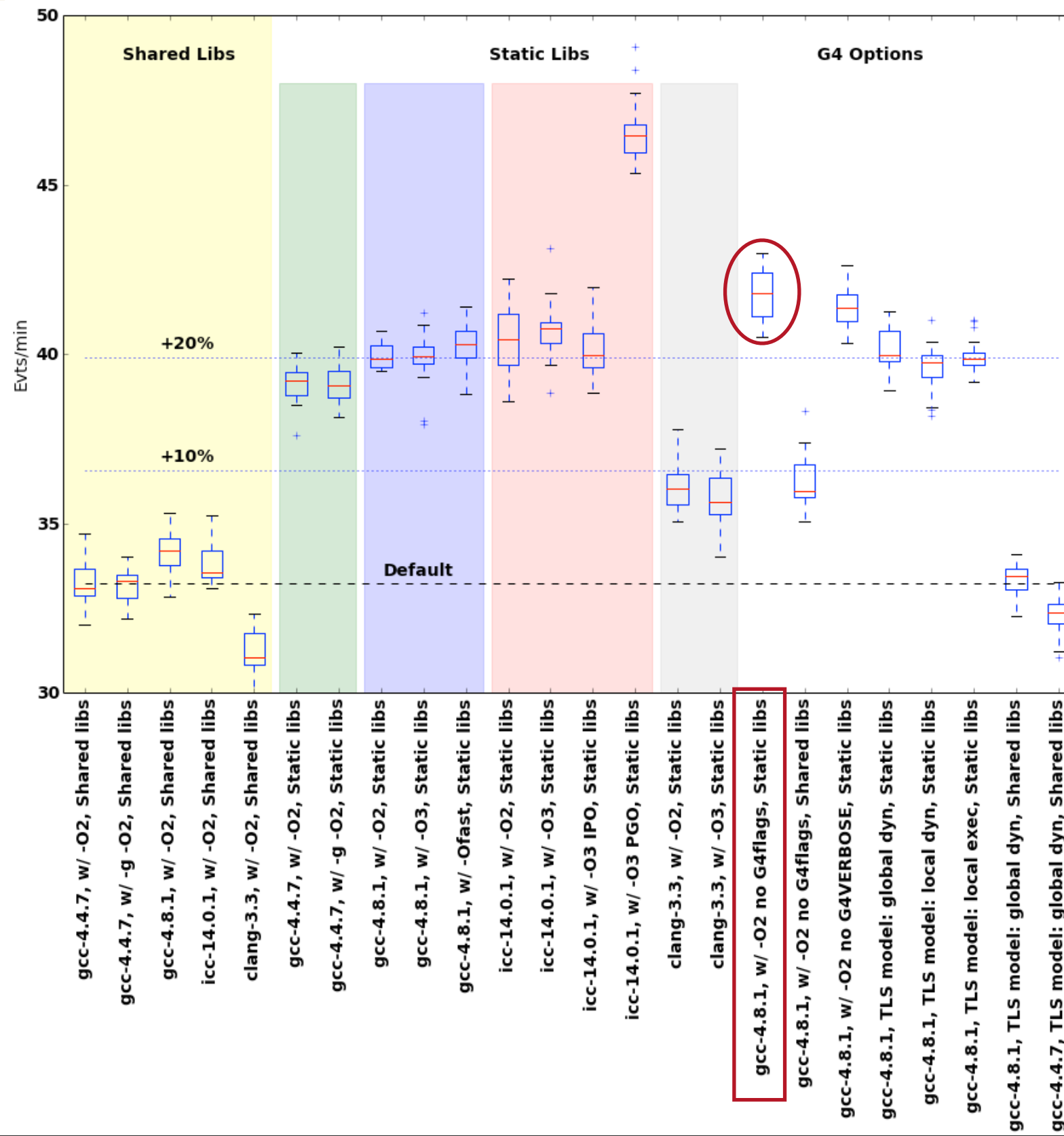
(***)Power consumption measured via “Intel Xeon Phi Coprocessor Status Panel”

All other are max TDP specifications

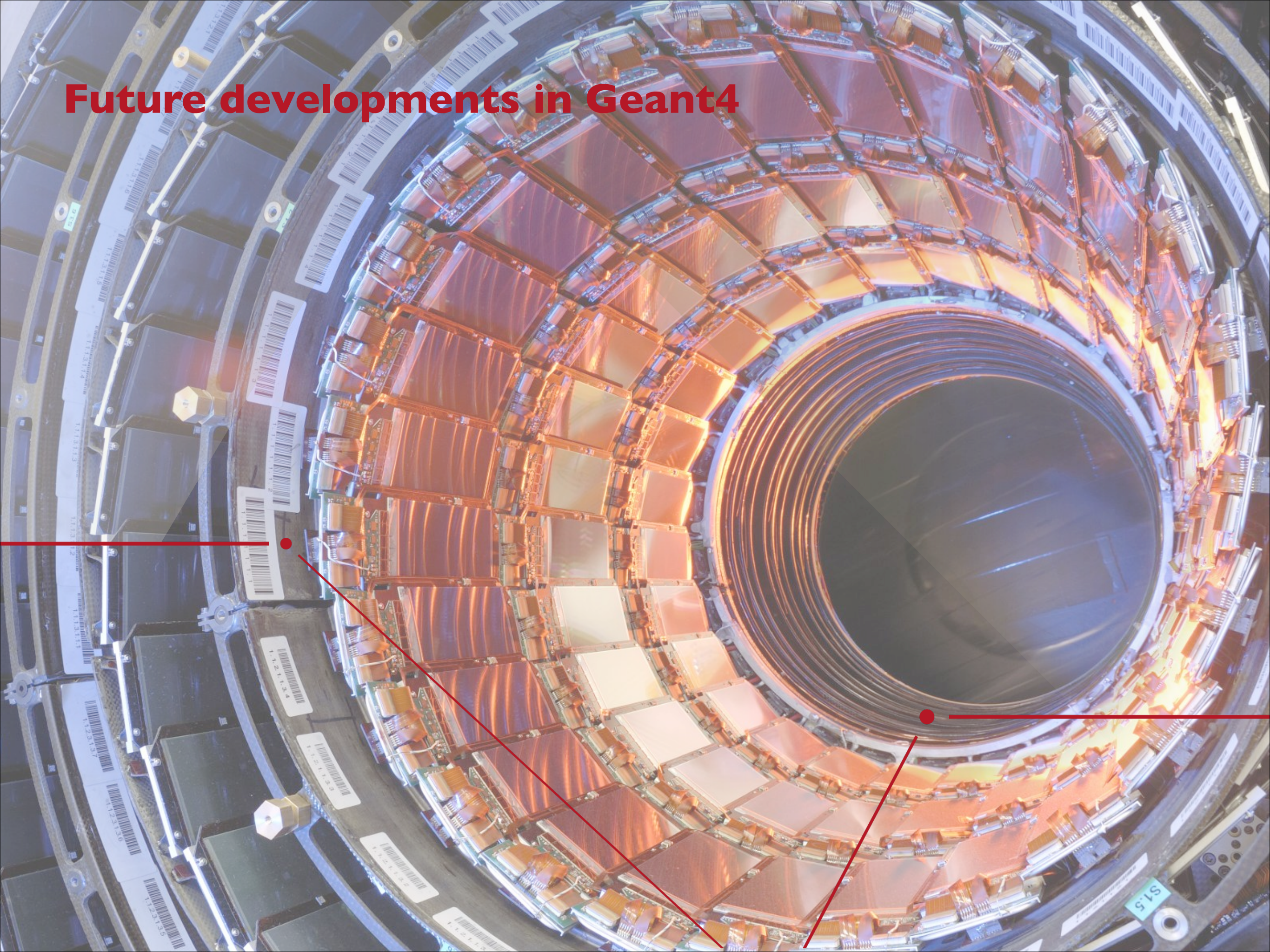
Comparison of compilation options/libraries



- Using to static libraries shows a 20% gain in performances
- Turning off some G4 options for productions also additional brings 6-7%



Future developments in Geant4



- **Further reduce memory consumption.** Rule of the thumb: fit complex simulations w/ $O(100)$ threads in $O(\text{GB})$ memory
 - e.g. typical computing power of accelerators
- In our experience: minimize memory usage can sometime **conflict** with other performance considerations (e.g. reduce memory “churn” via caching need special attention for thread-safety)
- Most memory consuming objects: geometry and EM physics
 - Efficient memory reduction already achieved in 10.0.beta
 - Next: need to **concentrate on Hadronics physics** (especially: cross-sections, specific models with large not-shared tables -BIC-)

Task-based model: CMS

- Current design assumes thread and worker are same thing...
- Not always easy to integrate with external frameworks based on **task concept** (no direct control of threads).
 - Strong interest from CMS on Intel's TBB, ATLAS is also considering it (at least as a level of study)
 - **Important requirement:** assume we have a pool of tasks of different nature (generation, simulation, digitization, reco, I/O) to be executed by a set of threads. We want to “occupy” only a fraction of the threads with simulation task at any given moment. This requires “migration” of simulation from a thread to another one (“clean up” is the difficult part)
- Introduced concept of “**workspace**”:
 - Encapsulate all thread/task private data in resource that can be exclusively requested, used and released
 - Currently only limited functionality for geometry module

“Splitting” of events: ATLAS

- ATLAS framework can already implement a **simple sub-event parallelism**:
 - Get a single generator event (hundreds primaries) and divide it (e.g. by region)
 - Each “piece” becomes a G4Event
 - Hand over to G4 each separately. Effectively split a huge HEP event in many G4Events
- Possible to use MT to parallelize
 - Once framework is made thread-safe: work ongoing

Heterogeneous parallelism: MPI based G4MT

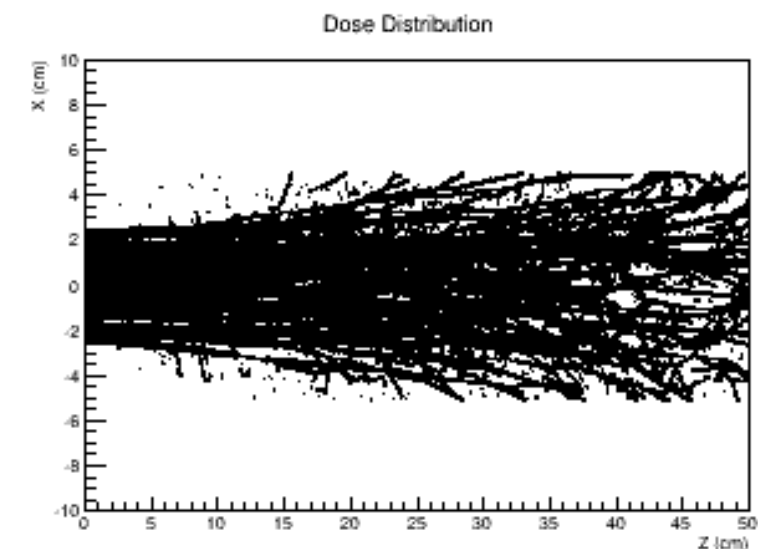
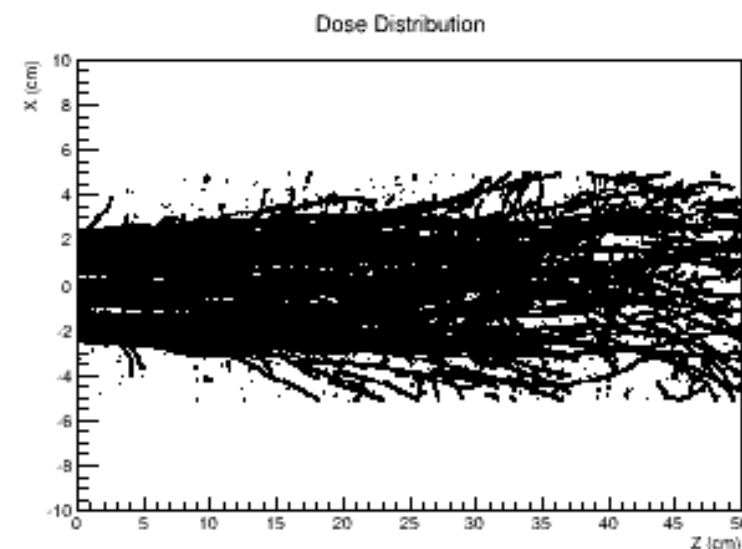
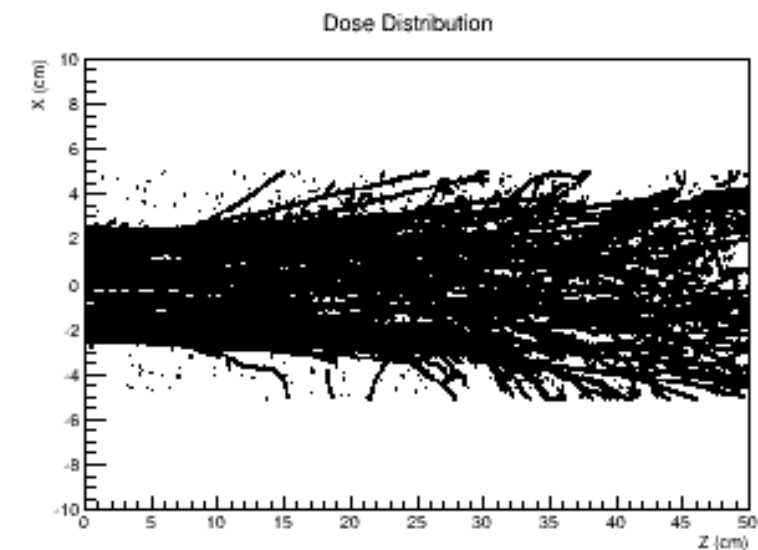
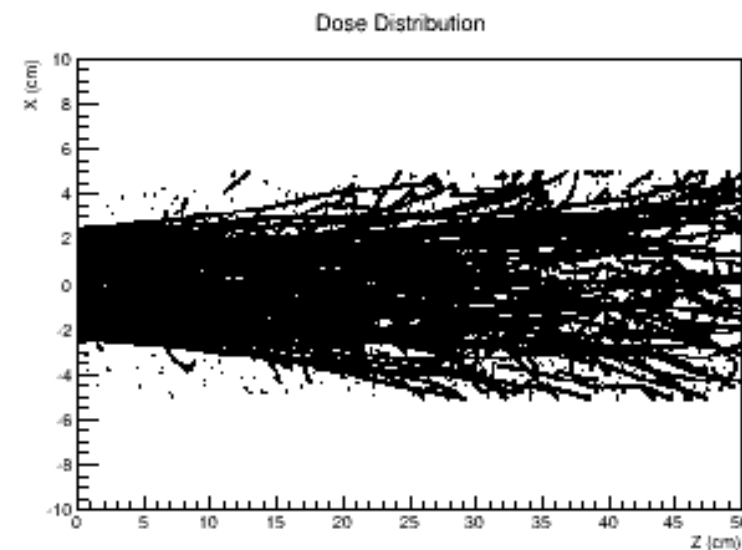
- **MPI based parallelism** available in Geant4
 - MPI works together with MT
 - Probably most interesting for non-HEP domains and/or SuperComputers

Example:

4 MPI jobs
2 threads/job
MPI job owns histogram

Next Step:

Host + MIC simulation
Based on MPI



2014+ simplified work-plan

- We already have **intra-node** and **intra-core** parallelism efficiently in place
 - Algorithm implementations are the place to look to get more performance (examples: see G4em review from Krzystof and Had XS caching from Pedro)
- From profiling analysis:
 - G4 profile **is very flat**: top 5 functions take: 3%, 2%, 2%, 1%, 1%
 - For HEP use cases CPU-time is spent equally in geometry and physics (~30% each)
- From profiling we can see what to improve:
 - Complex algorithms: often they are “**plain translations**” of complex physics formulas: not CPU efficient (but easier to read)
 - Review use of (large) **arrays and caching** of numbers (especially true for MT)
 - Introduce modern **parallel** RNG engine (RNG takes ~1%) and use of RNG vector interfaces
 - Similarly look at other “mathematical” aspects: 3- and 4-vectors
 - Switching to fast G4Pow and G4Log brought several % improvements

Gooda Example (ParFullCMS)



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	instruction_starvation	bandwidth_saturated	branch_misprediction	store_resources_saturated	instruction_latency	exception_handling
	8245843 (100%)	5703358 (69%)	3905966	4766639	2745332 (33%)	4665713 (56%)	80157 (0%)	522052 (6%)	204496 (2%)	1724700 (20%)	41728 (0%)
G4PhysicsVector::Value(do...	249801 (3%)	183440 (73%)	81361	96333	157004 (62%)	102099 (40%)	3478 (1%)	16477 (6%)		72405 (28%)	606 (0%)
G4ElasticHadrNucleusHE::H...	175220 (2%)	110935 (63%)	119668	177152	2405 (1%)	3021 (1%)	70 (0%)	884 (0%)	467 (0%)	123127 (70%)	1660 (0%)
G4Navigator::LocateGlobal...	190355 (2%)	141634 (74%)	59378	74117	144841 (76%)	101920 (53%)	1958 (1%)	9192 (4%)	7890 (4%)	25192 (13%)	586 (0%)
G4PolyconeSide::DistanceA...	134038 (1%)	80485 (60%)	99852	125190	14410 (10%)	10643 (7%)	1153 (0%)	6291 (4%)	20 (0%)	58860 (43%)	636 (0%)
G4SteppingManager::Define...	131792 (1%)	89575 (67%)	79845	87085	75913 (57%)	68719 (52%)	358 (0%)	7225 (5%)	3597 (2%)	15542 (11%)	407 (0%)
G4CrossSectionDataStore::...	121696 (1%)	78271 (64%)	73528	96325	73975 (60%)	36391 (29%)	258 (0%)	4541 (3%)	21048 (17%)	13356 (10%)	318 (0%)
CLHEP::MTwistEngine::flat...	107107 (1%)	60097 (56%)	89551	93810	30359 (28%)	41231 (38%)	179 (0%)	7622 (7%)		15185 (14%)	447 (0%)
G4ClassicalRK4::DumbStepp...	82552 (1%)	37933 (45%)	113510	131667	2723 (3%)	9948 (12%)	30 (0%)	2335 (2%)	6141 (7%)	17739 (21%)	199 (0%)
G4Navigator::ComputeStep(...	110755 (1%)	76064 (68%)	52331	68123	28749 (25%)	103898 (93%)	626 (0%)	9063 (8%)	298 (0%)	11041 (9%)	417 (0%)
G4VoxelNavigation::Comput...	113825 (1%)	80083 (70%)	50370	58626	62875 (55%)	55998 (49%)	745 (0%)	8238 (7%)	4074 (3%)	11538 (10%)	398 (0%)

Where do we spend time?

	8229714	(99%)
/data/adotti/new/lib64/libG4processes.so	2862844	(34%)
/data/adotti/new/lib64/libG4geometry.so	2488445	(30%)
/lib64/libm-2.12.so	879001	(10%)
/data/adotti/new/lib64/libG4tracking.so	487678	(5%)
/data/adotti/new/lib64/libG4clhep.so	307310	(3%)
/data/adotti/new/lib64/libG4global.so	355528	(4%)
/data/adotti/new/lib64/libG4track.so	299122	(3%)

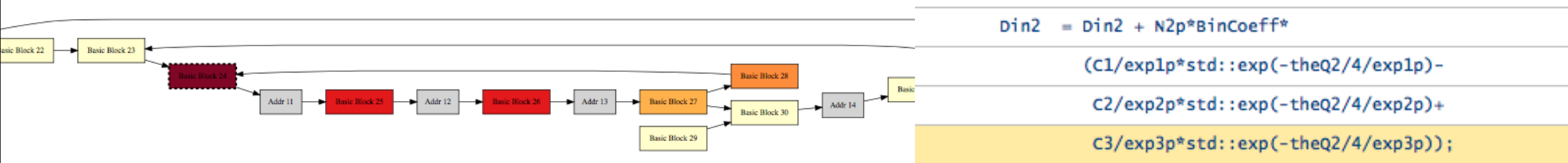
Physics
Geometry

B-Field

/da	8245843	(100%)
⊕ G4PhysicsVector::Value(double, unsigned long&) const	249801	(3%)
⊕ G4ElasticHadrNucleusHE::HadrNucDifferCrSec(int, int, double)	175220	(2%)
⊕ G4Navigator::LocateGlobalPointAndSetup(CLHEP::Hep3Vector const&, CL...	190355	(2%)
⊕ G4PolyconeSide::DistanceAway(CLHEP::Hep3Vector const&, bool, double...	134038	(1%)
⊕ G4SteppingManager::DefinePhysicalStepLength()	131792	(1%)
⊕ G4CrossSectionDataStore::GetCrossSection(G4DynamicParticle const*, ...	121696	(1%)
⊕ CLHEP::MTwistEngine::flat()	107107	(1%)
⊕ G4ClassicalRK4::DumbStepper(double const*, double const*, double, d...	82552	(1%)
⊕ G4Navigator::ComputeStep(CLHEP::Hep3Vector const&, CLHEP::Hep3Vecto...	110755	(1%)
⊕ G4VoxelNavigation::ComputeStep(CLHEP::Hep3Vector const&, CLHEP::Hep...	113825	(1%)
⊕ G4Mag_UsualEqRhs::EvaluateRhsGivenB(double const*, double const*, d...	109085	(1%)
⊕ G4SteppingManager::Stepping()	102486	(1%)
⊕ G4Transportation::AlongStepGetPhysicalInteractionLength(G4Track con...	99485	(1%)
⊕ G4PolyPhiFace::InsideEdges(double, double, double*, G4PolyPhiFaceVe...	81826	(0%)

Specific HAD σ
EM physics tables
Geometry navigation
Specific geometry
Navigation / σ interrogation
HAD σ
RNG
B-Field
Geometry navigation
Geometry navigation
B-Field
Geometry navigation
Geometry navigation
Specific geometry

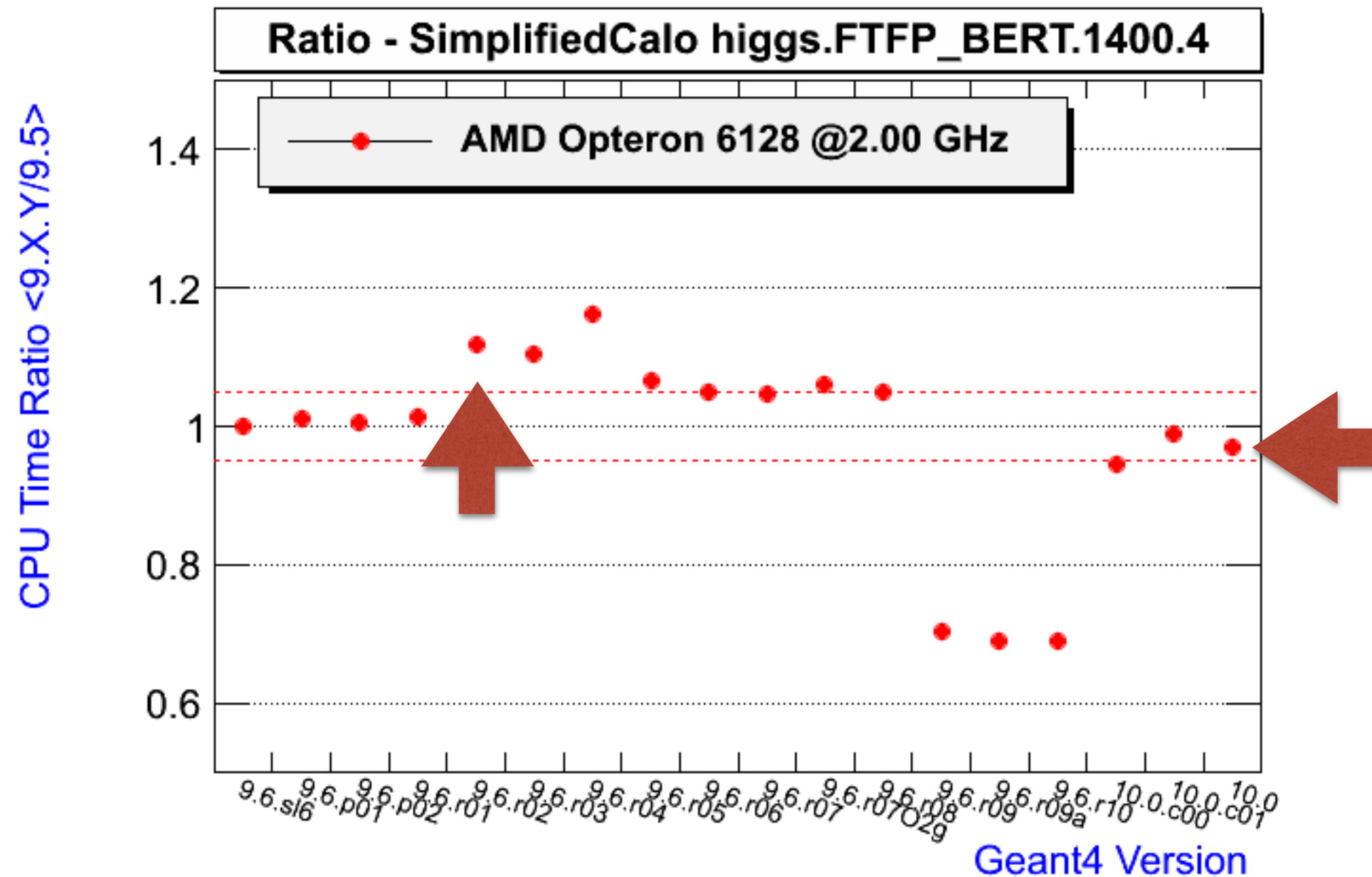
Example: our opportunities in the two top functions



Hadronic cross-sections (2%): two-level loop containing three `std::exp` and divisions between doubles



Benefits of MT developments for sequential code

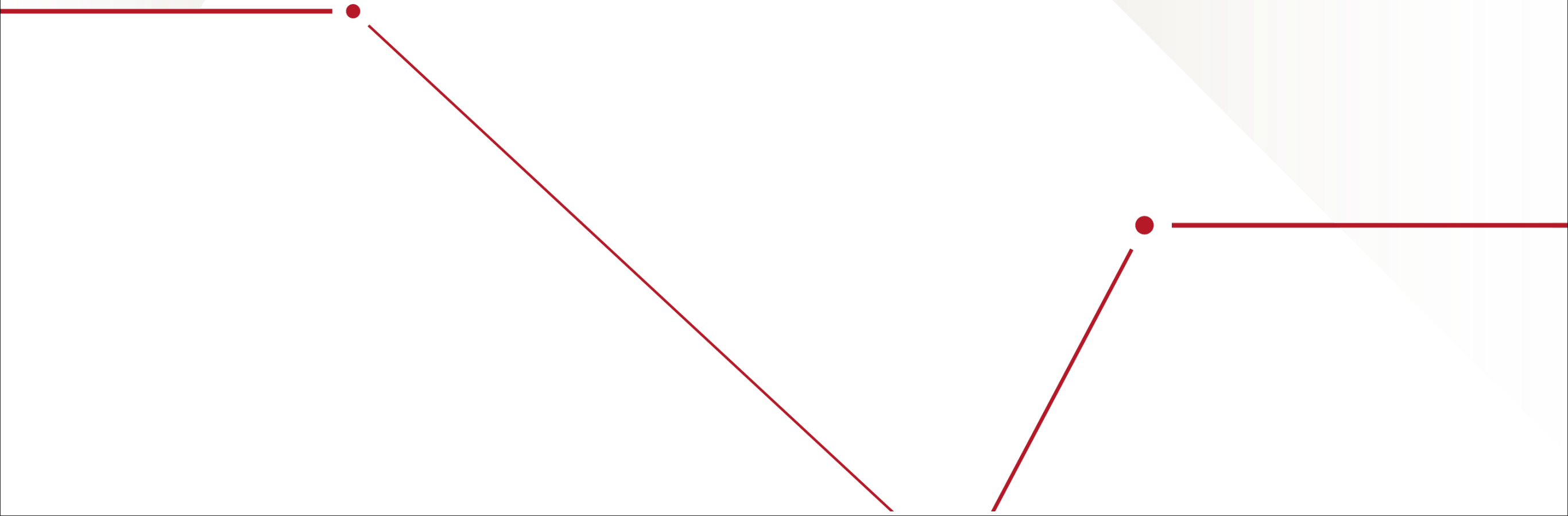


Very important lesson learnt this year: improvements in MT has **given benefits also to sequential users!**

Conclusions

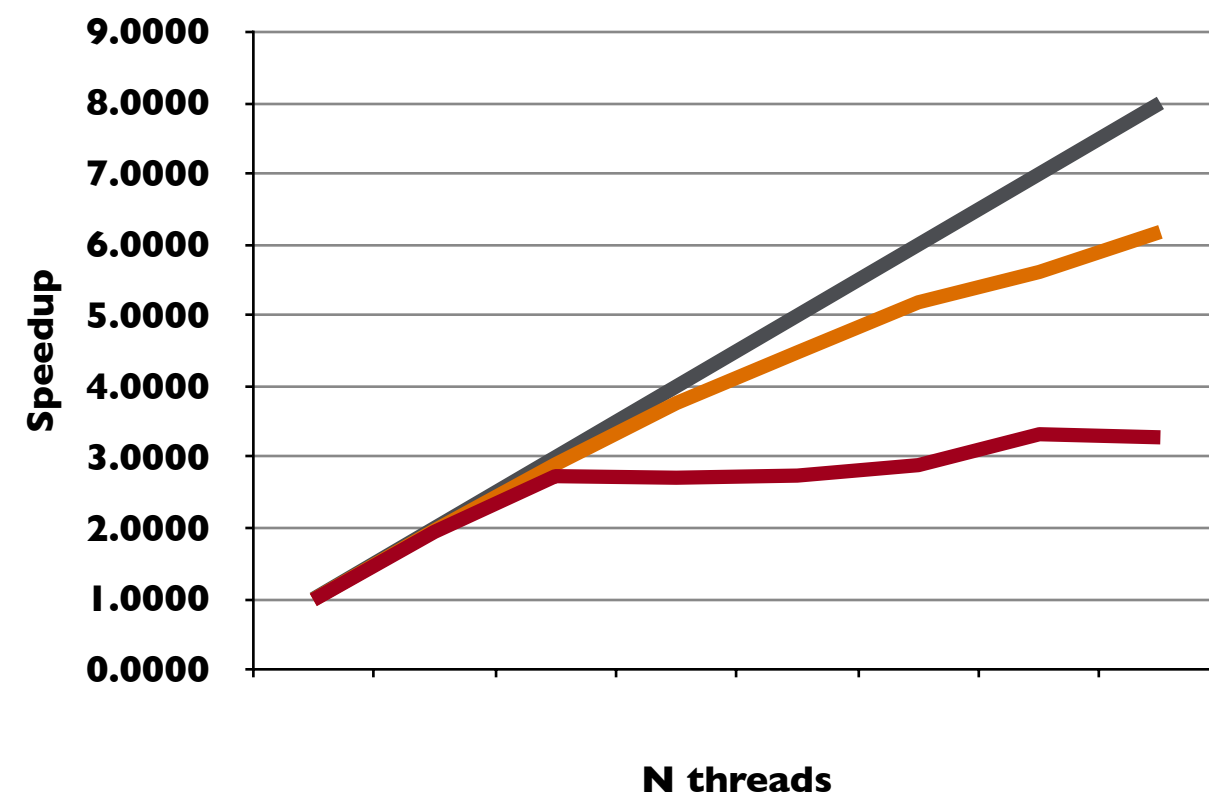
- Geant4 version 10 is the first large scale HEP software to **massively employ parallelism** through multithreading
- Thread-safe code is the **most important precondition** for parallel software: G4 code is now fully thread-safe
- Opportunities exists to speedup simulation (e.g. 20% from static libs) without need to modify code
 - We have seen clear trend in compiler generations in producing more efficient code. What else can we use better (e.g. PGO, auto-vectorization,...)?
- Large core-count and/or low-power consumption architectures can be used with HEP typical workloads
- Physics performance of G4 is demonstrated (e.g. Higgs discovery, treatment planning): improving the speed of physics algorithm would give greater benefit.

Backup



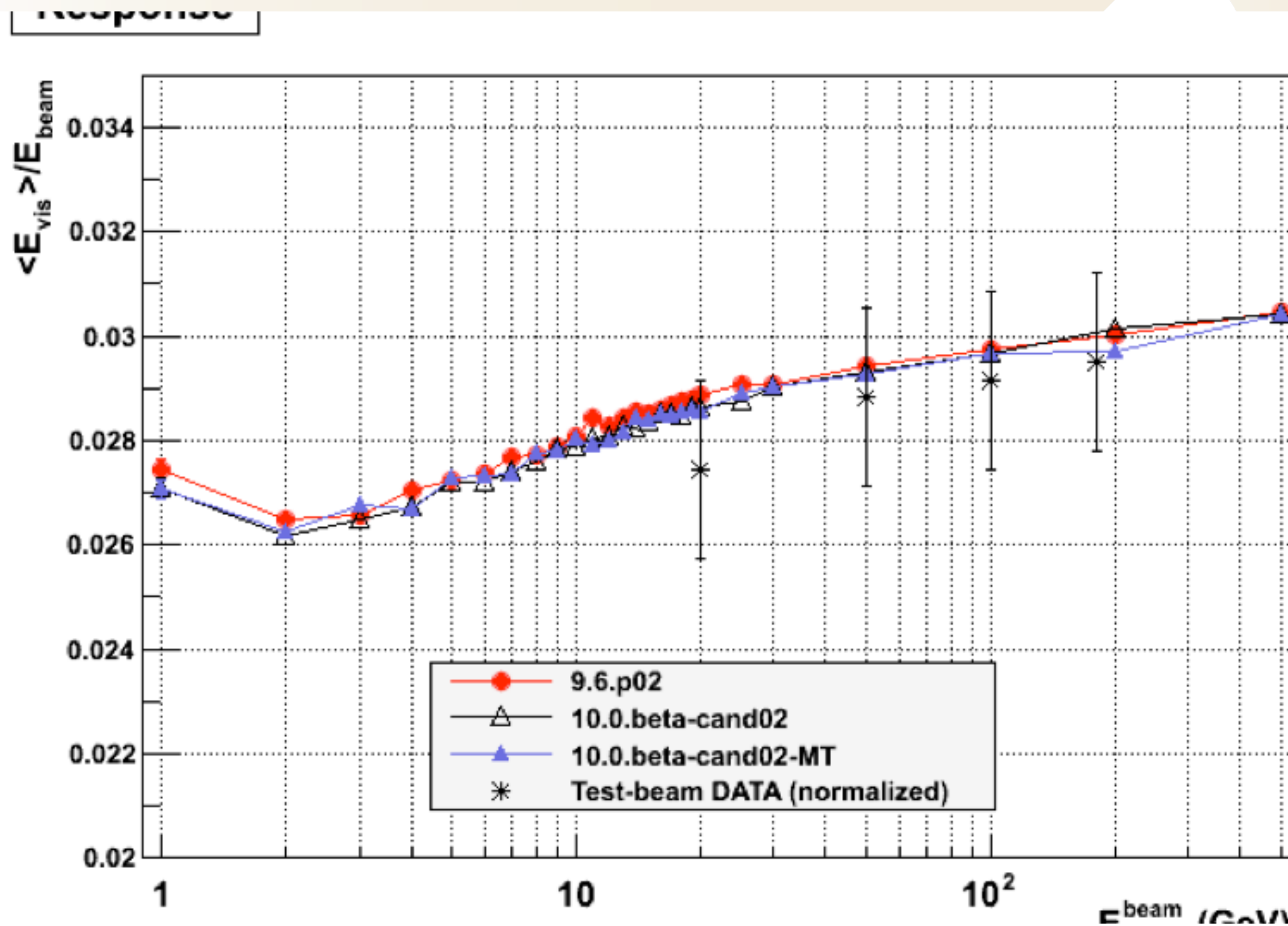
- Cross-compilation was relatively straightforward:
 - Adding **-mmic** to compilation flags via CMake
 - Modified couple of files implementing GNU specific pragmas
- Two ways of working:
 - Offload work to the card (via **#pragma**)
 - Native mode compile full application, start via ssh
- Only second option used so far: particularly attractive combining with MPI since minimizes data traffic over PCIe
 - Coordinate same program on host and card
 - Checkpointing successfully used to substantially speedup initialization - from $O(\text{min})$ down to $O(\text{sec})$
- Limitation was memory usage: only with top of the line MIC model can use all 244 threads

10% critical

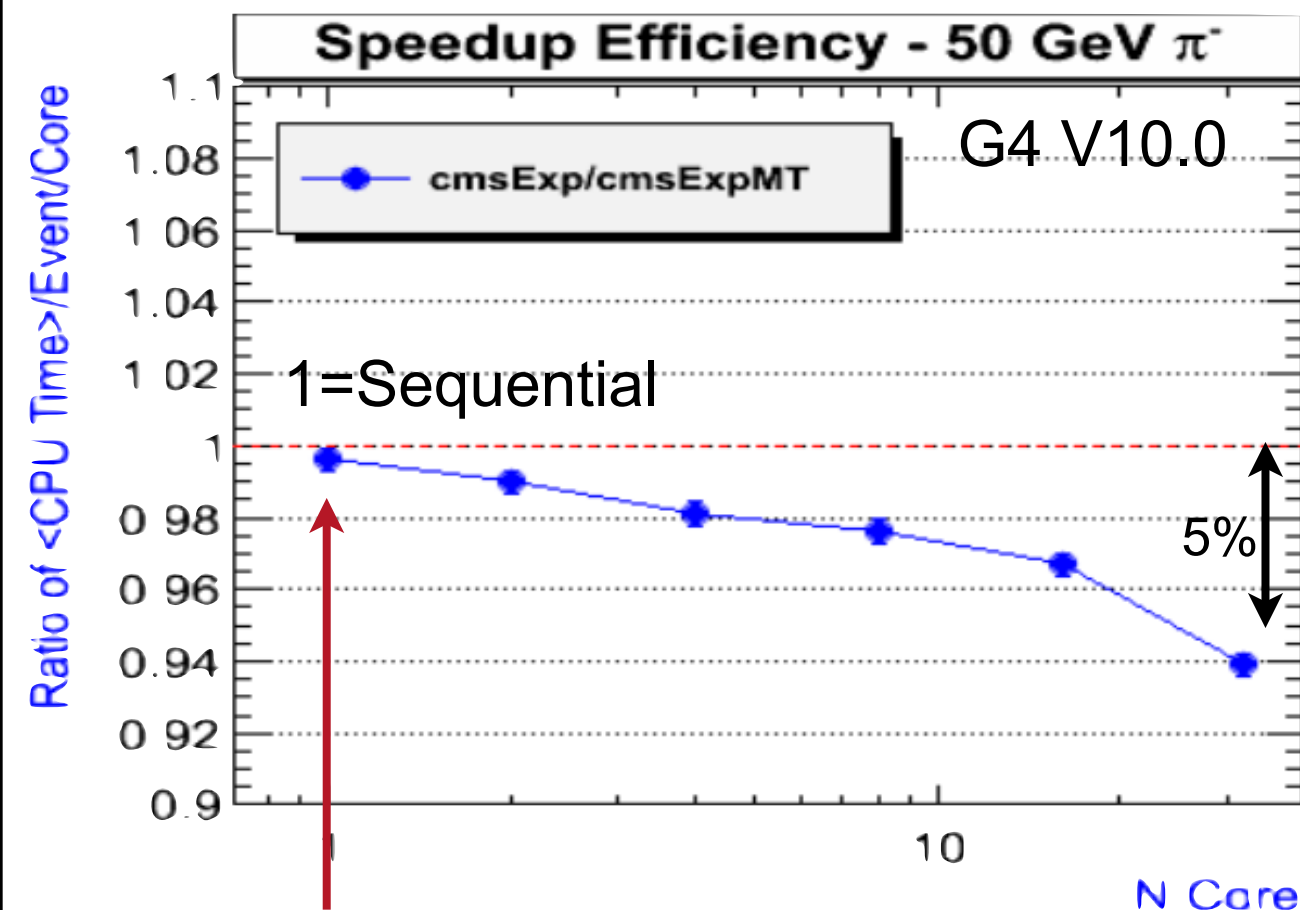


- Each (parallel) program has sequential components
 - **Protect access to concurrent resources**
- Simplest solution: use mutex/lock
- TLS: each thread has its own object (no need to lock)
 - **Supported by all modern compilers**
- Challenge: only simple data types for static/global variables can be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

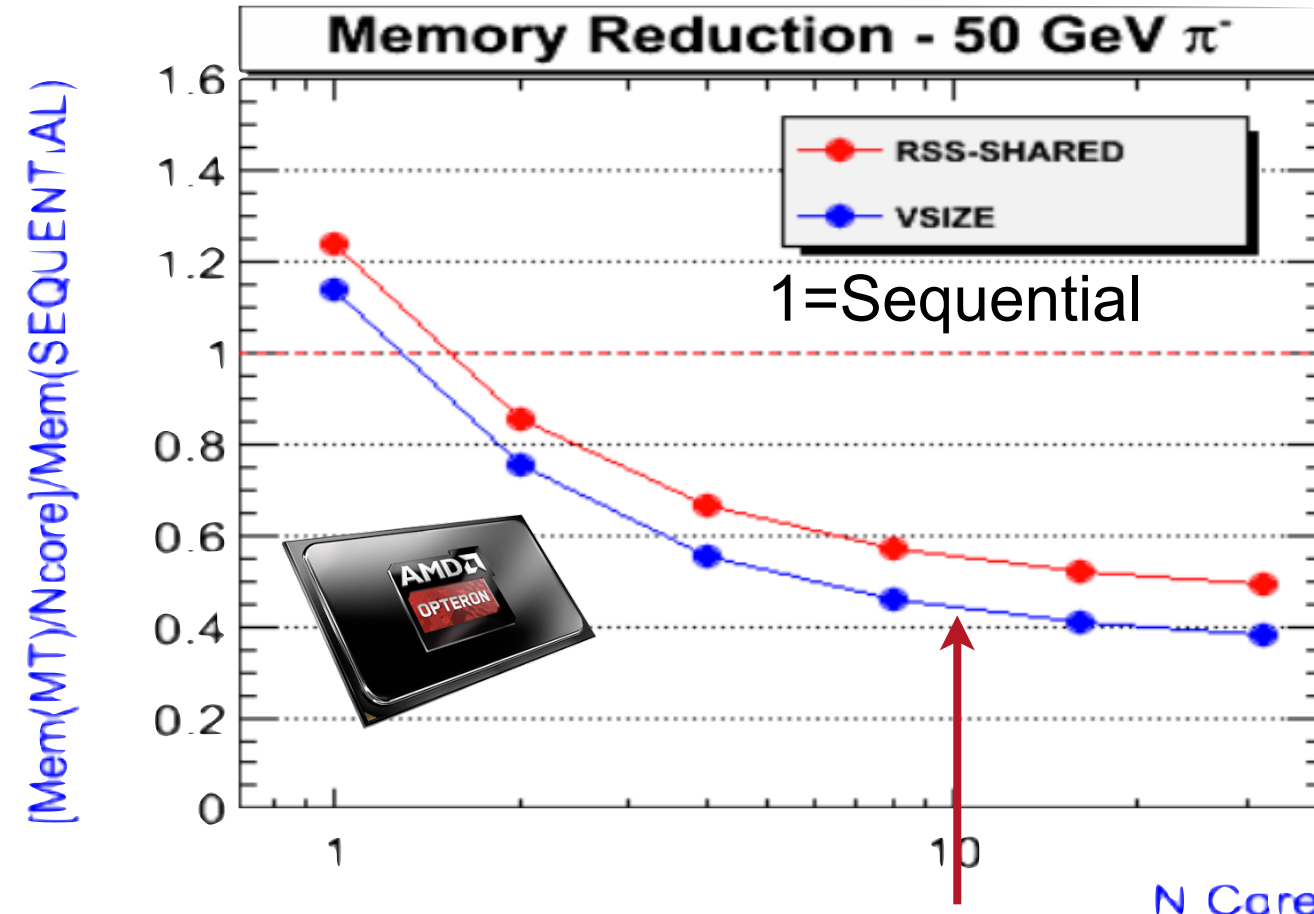
MT libs Vs SEQ libs



Comparing with sequential



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



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

Hadronic memory usage

- The hadronics most memory hungry (5MB) hot-spot is **BIC** model (even when not used). Some rework needed
- The second Hadronics components using more memory are **cross-sections** (2.2MB) stored in G4CrossSectionDataStore
- Models/processes account for about 1MB of memory
- It is realistic to reduce memory footprint for Hadronics of a factor 2
- Note: other models have a completely different profile
 - HP models: currently each thread load all HP tables, test11 for HP uses several GB of memory. No work on this done yet
 - Requires strategy for sharing database files