

Parallelization of ALICE simulation

a jump through the looking-glass

Matevž Tadel & Federico Carminati



- I. Introduction – why
- II. Workbook – what & how
- III. Results / comparisons
- IV. Conclusion

Intro – Motivation



1. Exploitation of many-core CPUs.
2. Code-review of simulation code.

□ **Just do it:**

1. How well does it work?
2. How hard is it - what skill level is required?
3. What are required changes in AliRoot and externals?
4. Make an informed plan for further work.

□ Simulation a good candidate:

- No input and little output – evaluate CPU usage.
- Simulation code not expected to change.

□ **Mission: Parallelize simulation with Geant3 as far as needed to have a full picture.**



Intro – ALICE simulation facts



- Transport engines:
 - **Geant3** – main engine, used for all productions
 - **Fluka** & **Geant4** interfaces basically done
- Geometry
 - **TGeo** used everywhere (also in reconstruction)
- Implemented as *Virtual MonteCarlo* (VMC) appl.
 - All geometrical queries done in **TGeo!**
- Simulation job for a central **PbPb** event:
 - # primaries: **~65k tracked**
 - time: **~5h** [x2 for F & G4]
 - output: Kine: **26M**, Hits: **274M**
 - memory: **572M (1.2G virtual)** [~same for F & G4]

Workbook – What we did

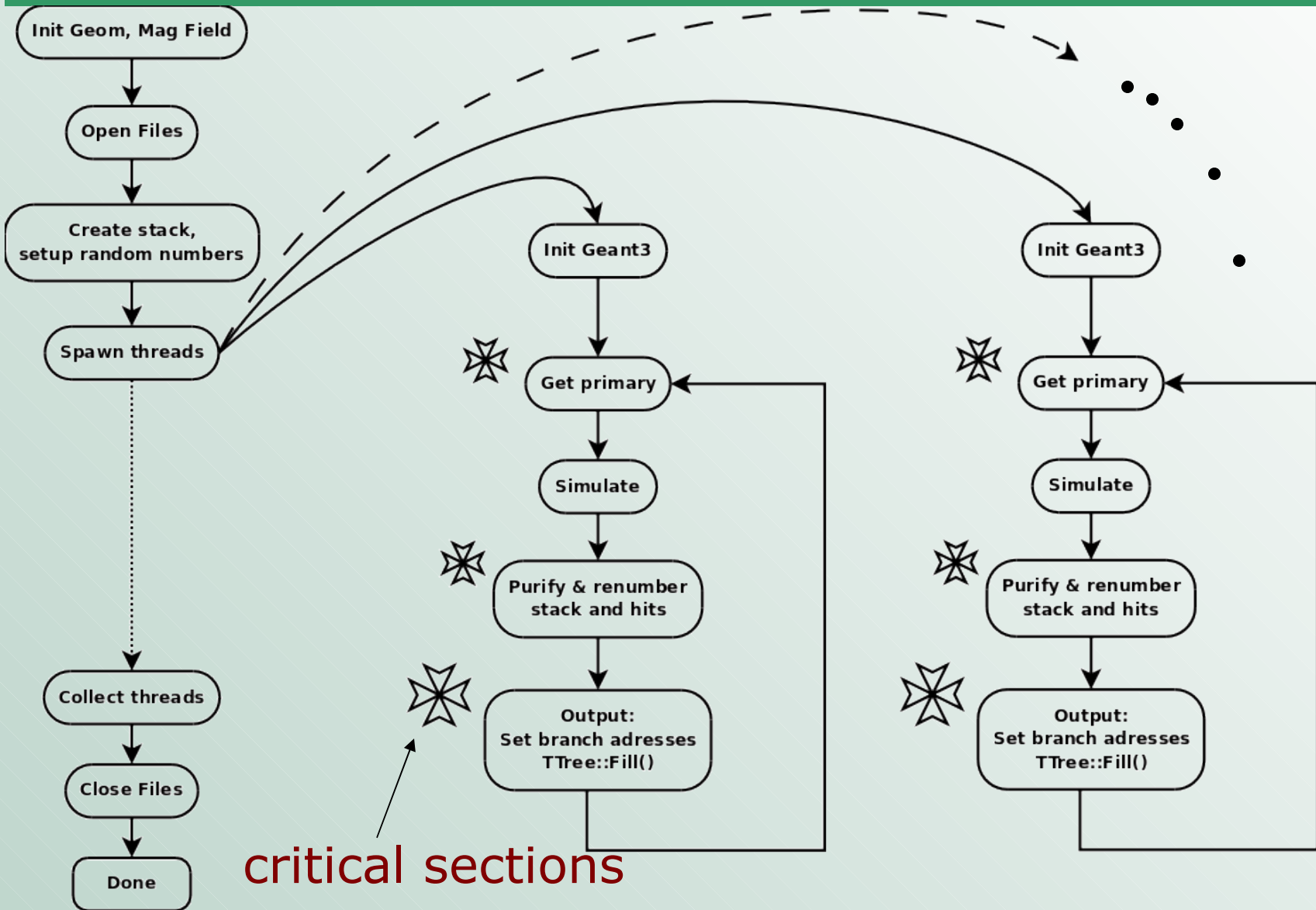


- **Geant3** made thread-safe:
OpenMP => all commons and static data made *thread-private*
 - **TGeo** made thread-safe: *TGeoNavigator* was already there.
Introduced thread-specific structures for five classes.
 - Implemented a **VMC** application supporting tracking of several concurrent primaries in dedicated threads.
 - Simulation threads take primaries from global stack.
 - Data is written out after each primary.
- Not a big change to switch to event/job-level parallelism.

Tools:

- *gcc-4.3 branch* ~4.3.4
- *OpenMP* *limited usage (threadprivate pragma)*
- *Posix threads* *in particular thread-specific data*
- *Computer* *4-core x86_64, 4GB RAM, linux-2.6.27 64-bit*

Workbook – Process scheme



Results – Correctness testing



Use simple geometry from G4-VMC/Example03.

- Make sure one gets identical results (***which we do***).
- Use 1 TeV ***e***, ***p*** and ***n***.
- For hadronic processes:
 - increase EM cuts to get **~75% of hadronic secondaries**;
 - Otherwise EM processes dominate.

For correctness testing:

- particles were accumulated till the end (*no output*)
- hits were not stored (*not much memory allocation*)

For 4 cores: **3.92 speedup** (*including initialization*)

Results – ALICE **disclaimer**



We used ALICE geometry, but **not full AliRoot**:

- Using AliRoot would only affect step-sizes and cuts.
- Use a single step-callback / hit processing function.

The base-line memory is **larger** for AliRoot:

1. Resident: **570 MB** [180 MB in further results]
2. Virtual: **1200 MB** [320 MB in further results]

This includes **all** memory usage by a single threaded process: code, containers, IO buffers.

Memory usage per thread is realistic!

Results – Initial memory usage



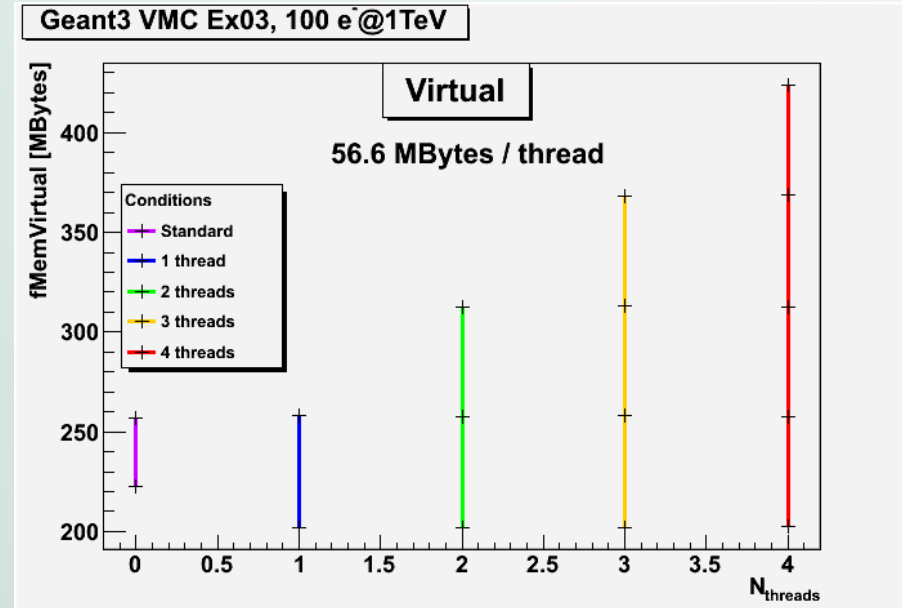
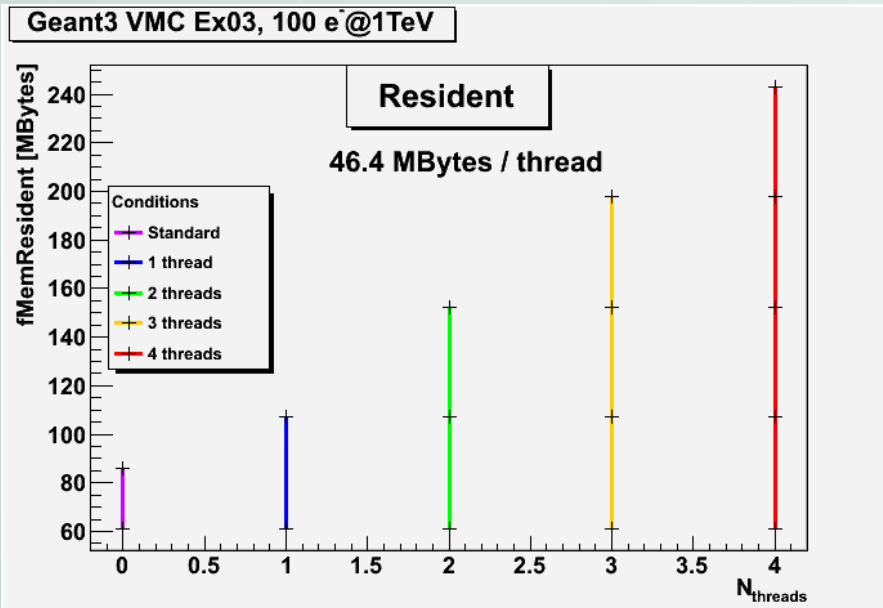
Initialization of MC-threads – basically Geant3.

Could be further reduced (*share cross-section data*)

- Separate shared part of Geant3 commons – init them once.
- KSM

This is a start - we will also get:

1. particle-stacks & hit containers [TClonesArray per primary]
2. IO buffers.



Results – ALICE – no output I.

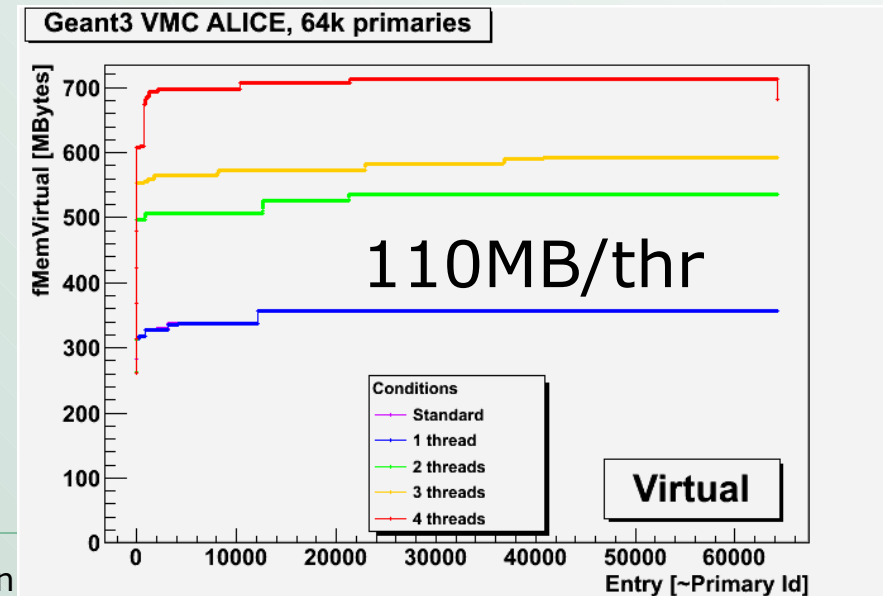
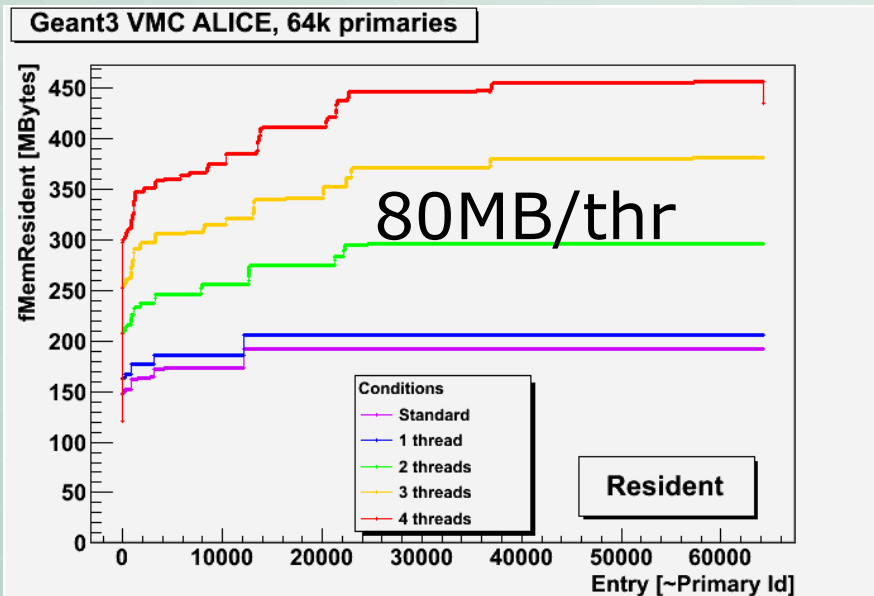
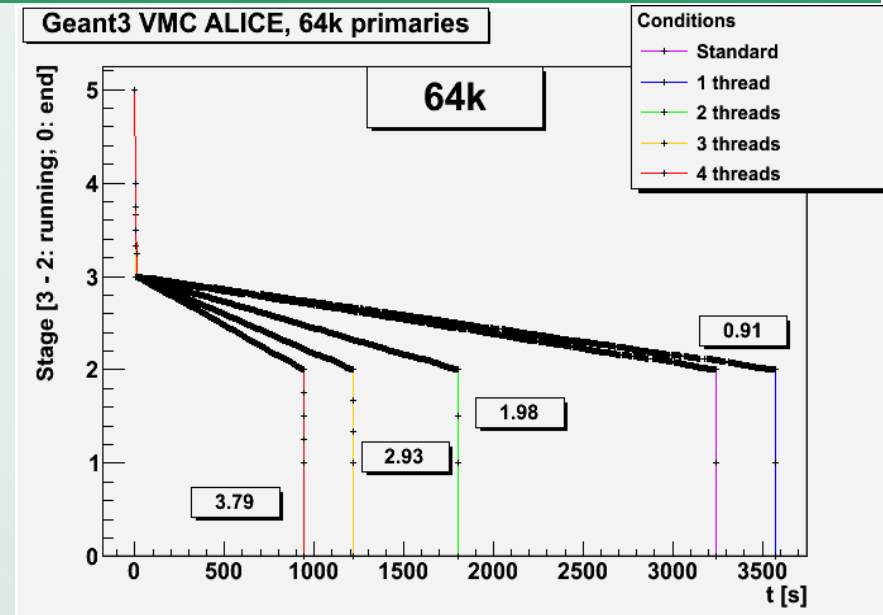


Memory: +35MB/thr [65v]

Stack & TGeo thread-data

Speed:

1. **9% loss** - see next slide
2. Scaling slightly degraded:
 - a) Realistic geometry
 - b) Uneven # of secondaries



ization

Results – ALICE – no output II.



Callgrind on tracking of one 1 TeV e⁻:

- 20% spent in `__tls_get_addr()` [**with Ex03 geom**]
- **every** access to a common variable goes via this!

Complained to *gcc* mailing list, proposed to get the base-address once at each function entry.

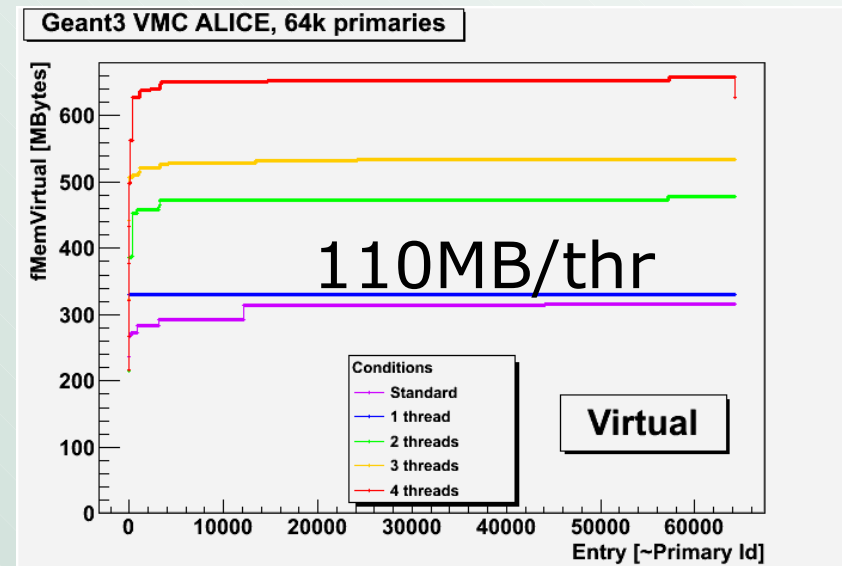
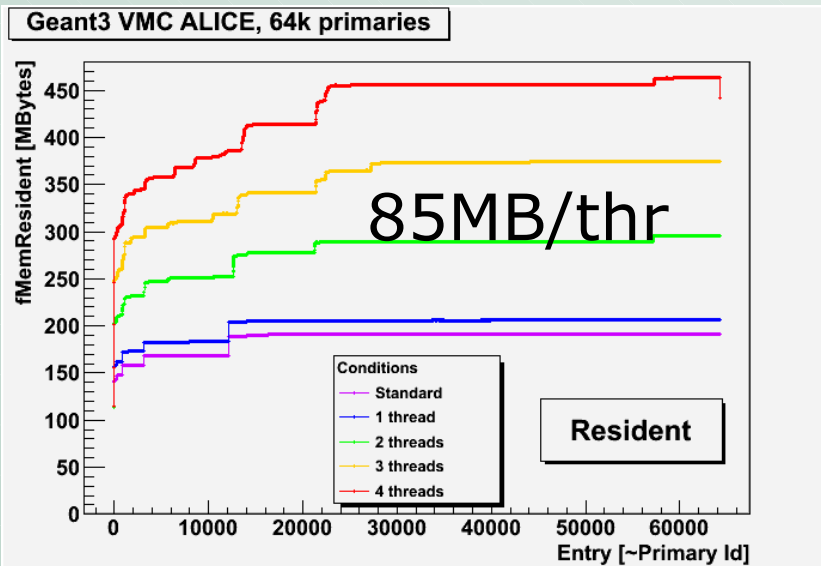
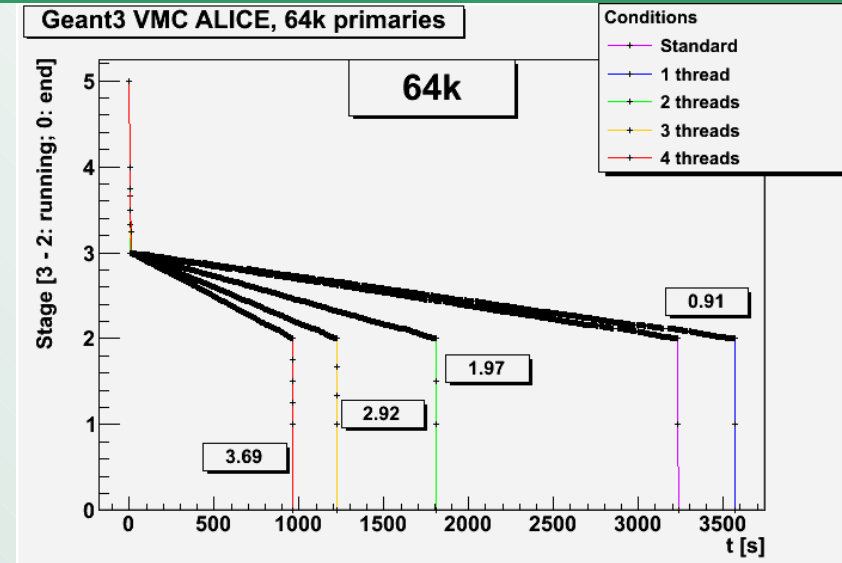
Still an open issue. If fixed, loss would only be **1%**.
estimated from # of calls and # of function entries

We were told to use static linking – that there is no penalty there. Still on our *ToConsider* stack.

Results – ALICE – 300MB output



- Scaling for 4 threads:
3.69 (down from 3.79)
Locking during output.
- Memory: +5MB/thr [0v]
 - TClonesArray for hits
 - IO buffers for particles/hits



Results – ALICE – 2GB output

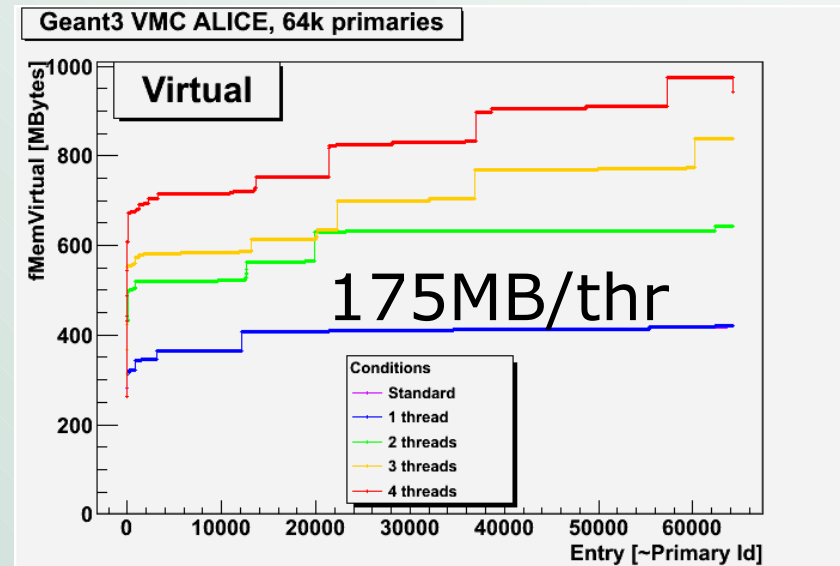
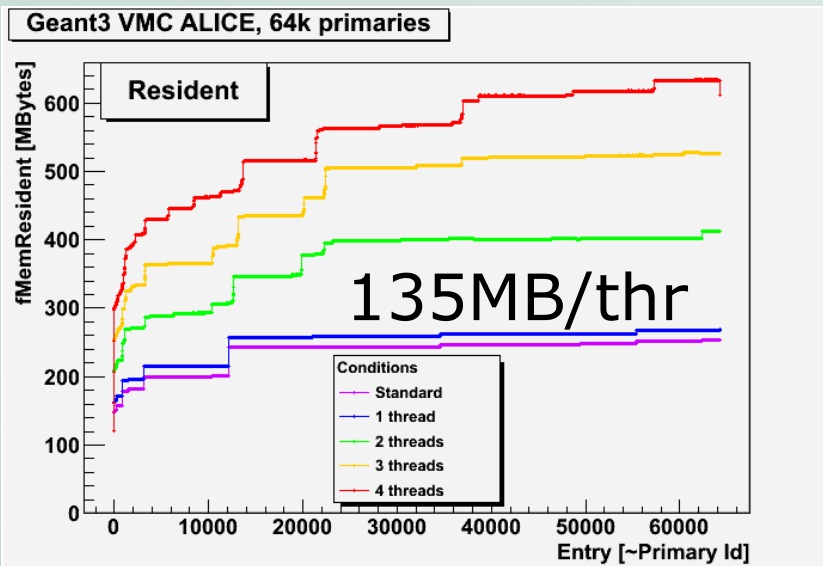
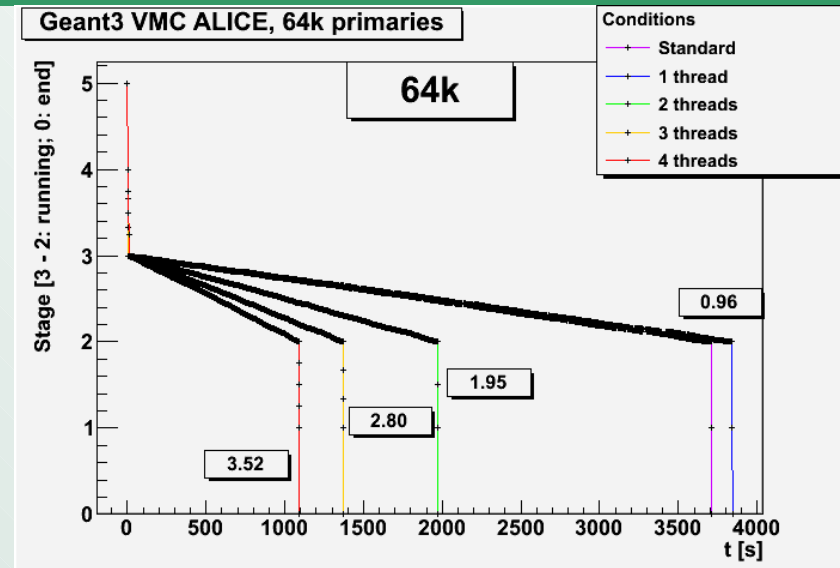


Check impact of more IO.

Compression: **15%** more CPU

1. Less difference for 1 thr.
2. Poorer scaling: sequential IO begins to bite

Memory increases: larger IO buffers.



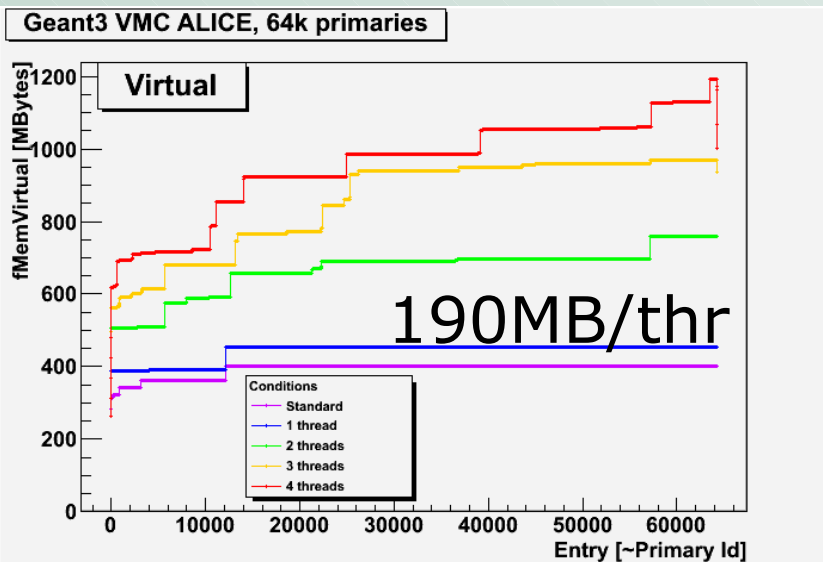
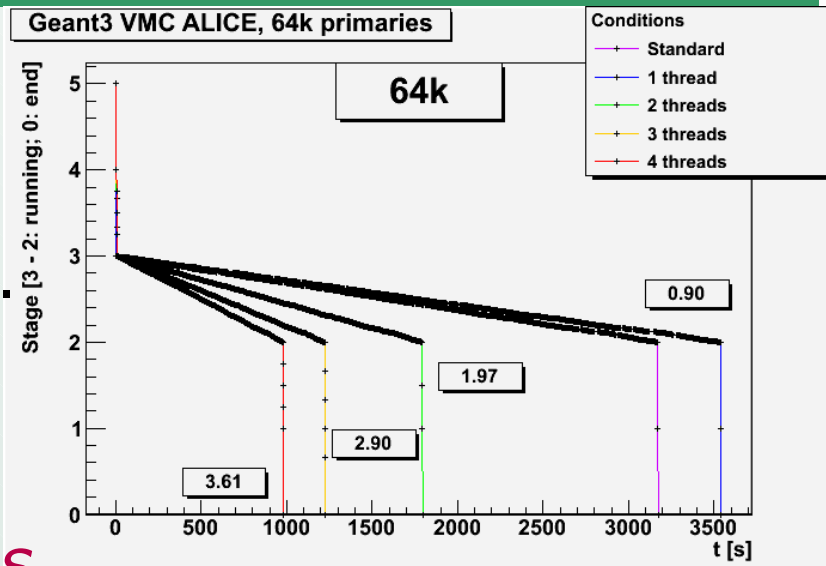
Results – Output in a dedicated thread

Avoid waiting for serialization!

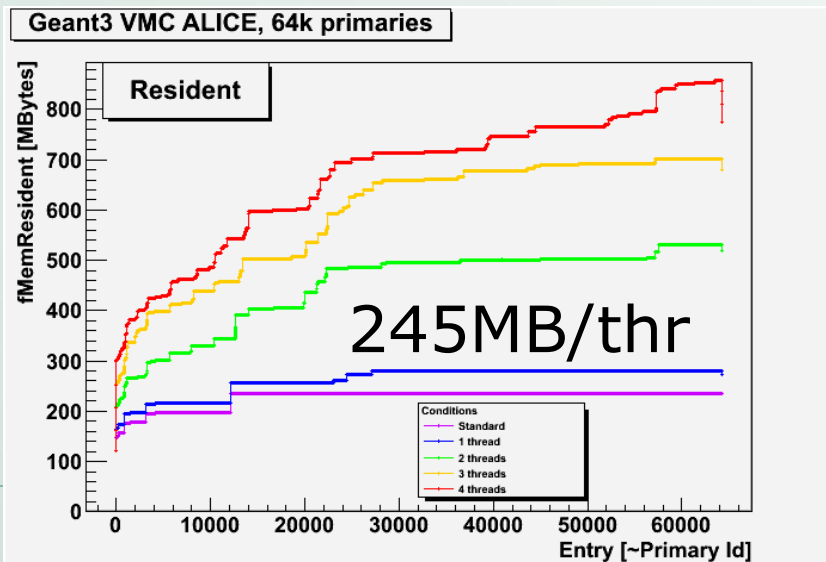
- Particle/hit arrays created as needed [eg. ~20 for 4 threads]
- Memory usage per thread doubles.
- As arrays get passed among threads, speed is affected, too.
- Limit # of containers per thread?

The solution: *Fill and compress output buffers in the MC thread!*

[with output size 300MB]



parallelization of



Results – Realistic comparison



Compare under same CPU load (*300MB output*):

1. 4 single-threaded jobs running concurrently
2. 4 four-threaded jobs running sequentially

Wall-time-ratio: 0.942 [would be 0.923 taking 3.69 speedup]

With parallel thread initialization becomes **0.976**.

Memory-ratio: 1.77 [1.93 virtual]

For AliRoot one should add **+400MB** [+900MB virtual] per process:
ratio becomes **2.8** [3.0 virtual]



ization

Conclusion



We consider this a successful attempt:

- We have a testing / benchmarking framework for parallelization of simulation.
- Memory usage per thread $\sim 100\text{MB}$ – very good!
- Scaling – issues, but satisfactory.
- Problems are part of the game – start early!

Memory buffers and IO are crucial.

For track-parallelism need thread-safe way of filling *TTrees*.

Structure of code improves with thread-safety:

- clear separation between **local** and **task** data;
- task context naturally becomes thread-private.



- Open issues:
 - Follow-up on thread-private data access in OpenMP
 - Try **KSM** and **HOARD** memory allocator
 - Try running on machines with more cores
- Plans:
 - Push thread-safe **TGeo** into **ROOT**
 - Consider event-level and job-level parallelization
 - Consider other transport engines – with VMC!
 - Explore **VMC** parallelization towards a final solution
- Use all this with **ALICE** simulation
 - When issues resolved and ground-work finished
 - Digitization – memory usage goes to 2GB virtual!





- **Geant3** thread-safety
 - Use OpenMP to make all **commons** and **saved** variables thread-private
 - Issue: **equivalence** between thread-private variables not allowed by OpenMP specification
 - Requested extension - now an open issue in gcc tracker

- **TGeo** thread-safety
 - Most work already done by the author (**A. Gheata**)
 - Class TGeoNavigator – contains tracking state & stack
 - A couple of small structures made thread-private
 - Voxelization and division – *navigation optimizations*
 - Composite shapes and shape assemblies – *state info*



- As VMC is virtual, it is hard to parallelize ☺
 - **TGeant3** and its subclass **TGeant3TGeo**
 - Example03 from Geant4 (simple Pb calorimeter)But general concepts could go into the base classes.

- **TMContext** – encapsulate tracking state for each primary:
 1. MC transport engine (*TGeant3TGeo*)
 2. Particle stack and Hit container
 3. Random generator (*seeds set per primary*)
 4. TGeoNavigatorStored as thread-specific data.

Workbook – Running scheme



1. Initialize geometry and magnetic field
2. Initialize worker threads:
 - N transport threads (nice 20)
 - **optional** – dedicated output threadCan be done in parallel ... left like this for monitoring.
- Run the threads - *feed them from top-level primary stack*
 1. Get primary, setup random seed
 2. Process with inner MC loop
 3. Output particles and hits:
 - Without IO thread:**
 - TBranch::SetAddress() – for particles and hits
 - TTree::Fill() – fill the trees, compression is also done here
 - With IO thread:**
 1. Pass TClonesArray's to output queue
 2. Acquire a new set of arrays from a pool (or create new ones)



- We didn't run these tests in full AliRoot.
- In particular, this avoids the detector-specific code for Hit processing and setting of step-size.
 - Not much would have to be changed there:
Get virtual MC and Hit arrays as arguments
(from context)
Some static variables need to be removed.
95 files would need to be modified.
 - The detailed changes will depend on the final form of parallelized VMC and TGeo.
- But we were running with full ALICE geometry.