

# FLUKA.CERN STATUS & PLANS SCORING

---

Gabrielle HUGO on behalf of FLUKA.CERN collaboration

Wednesday, September 28<sup>th</sup> 2022

# FLUKA.CERN STATUS & PLANS

---

# A BIT OF HISTORY

---

- FLUKA was **born in the 60s at CERN** with Johannes Ranft, who wrote a **set of tools for high energy proton accelerators shielding design**. Among these tools, the first cascade code was called FLUKA (FLUktuierende Kaskade).
- Further developed in the **70s** for the SPS.
- Tools **unified in the 80s** into a single hadron cascade code, with **flexible geometry and a modern formulation of the hadron interaction model** (Leipzig University, CERN, Helsinki University).
- In the **90s: multiparticle, multipurpose code**. **SSC and LHC design needs** lead to transforming FLUKA from a high-energy code mostly devoted to radiation shielding and beam heating, into a code which could handle most particles of practical interest and their interactions over the widest possible energy range (INFN).
- From **2003 until August 2019**: maintained and developed under a CERN & INFN agreement, **reaching more than 10,000 users**. Support for ions interactions, creation of FLAIR, new neutron cross section library below 20 MeV (including 260 neutron and 42 gamma groups), improved models...
- **From December 2019**: INFN and CERN separately continue the development, maintenance and distribution of FLUKA. New CERN distribution aiming to ensure FLUKA's long-term sustainability within an **international collaboration**.

# FLUKA CAPABILITIES

---

- **FLUKA** is a general purpose Monte Carlo code for the interaction and transport of hadrons, leptons, and photons from keV (with the exception of neutrons, tracked down to thermal energies) to cosmic ray energies in any material.
  
- **Capabilities:**
  - hadron-hadron and hadron-nucleus interactions
  - nucleus-nucleus interactions (including deuterons!)
  - photon interactions ( $> 100$  eV)
  - electron interactions ( $> 1$  keV; including electronuclear)
  - muon interactions (including photonuclear)
  - neutrino interactions
  - low energy ( $< 20$  MeV) neutron interactions and transport
  - particle decay
  - ionization and multiple (single) scattering (including all ions down to 250 eV/u)
  - coherent effects in crystals (channelling)
  - magnetic field, and electric field in vacuum
  - combinatorial geometry and lattice capabilities
  - voxel geometry and DICOM importing
  - analogue or biased treatment
  - on-line buildup and evolution of induced radioactivity and dose
  - built-in scoring of several quantities (including DPA and dose equivalent)
  
- **Applications:**

✓ Accelerator design	✓ Radiation damage	✓ Particle physics	✓ Neutronics
✓ Radiation protection (shielding design, activation)	✓ Radiation to electronics effects	✓ Neutrino physics	✓ ADS systems
		✓ Cosmic ray physics	✓ Medical applications

# PRESENT FLUKA LIMITATIONS

---

Present FLUKA results have reached a **high level of maturity**.

However, the code-base is mostly following **FORTRAN 77 standard**:

- 'Cryptic' code readability (all variables and functions described with up to 6 characters only).
- Extensive use of COMMON and shared variables.
- No clear dependencies (when and where is each variable modified?).
- Implicit variable declarations (hence any typo in variable name creates a new different variable, silently used).
- Static memory allocations.
- Lack of modularization, rigid code structure.

In addition:

- Locals are initialized to zero through compiler option (hiding uninitialized variables)
- Obscure error messages.
- Code duplications, dead code, etc.

Major issues for:

- **debugging & long-term maintainability**
- **further implementations**
- **openness to external contributions**



# FLUKA++: POSSIBLE PATHS

---

A complete transition to another MC was not an option due to:

- **long investment** in FLUKA (models, auxiliary tools, ...)
- **continuity of the physics results**

Several paths were envisaged:

- **Continue and evolve the present code**
  - \* **already operational**
  - \* **arriving to its limit** → adding new features becomes more and more complicated
  - \* **requires deep understanding of the code** → not easy for external contributors
  - \* **license restrictions**
- **Modernize its architecture using existing codes/libraries**
  - \* **progressive transition**
  - \* **establish synergies with other codes**
  - \* **combine with other models if more performant**
  - \* **add external dependencies**
  - \* **transition phase with work in multiple codes**
- **Re-write new code from scratch**
  - \* **very appealing to make a clean design from start**
  - \* **would be a long endeavor to arrive to the same performances**
  - \* **transition phase with work in multiple codes**



# FLUKA++: CHOSEN PATH

The **hybrid solution was chosen**:

- Build as a **Geant4 C++ application** (with G4 physics):  
general design, I/O, mathematical libraries, field tracking...
- Build **on top the core technical infrastructure**, needed to provide same features as FLUKA:  
geometry, materials, variance reduction techniques, magnetic fields, scoring.
- FLUKA physics models are progressively incorporated, until equivalent or better physics performance is reached.

## FIRST STEP: MOIRA

MOIRA: G4 application, supporting FLUKA (and G4) input files, access both G4 and FLUKA physics.



## SECOND STEP: RESULTS EQUIVALENCE WITH FLUKA IS REACHED RENAME MOIRA INTO FLUKA++

FLUKA++: G4 application, supporting FLUKA input files, access FLUKA physics.



# FLUKA++ VERSUS GEANT4: DIFFERENT PHILOSOPHIES

---

Different goals at the service of different use-cases:

- **G4:** Modular toolkit, allowing users to **develop applications** relying on a **common core base**.
- **FLUKA++:** Fully integrated application, directly used and benchmarked for the design/construction/operational studies of accelerators over several decades.  
Continuity in user experience with FLAIR (support huge number of **existing complex simulation setups**).  
Continuity in physics results with FLUKA4.



# MOIRA & FLUKA++: BENEFITS

---

MOIRA: Supports both G4 and FLUKA physics models from same application.

FLUKA++: 5<sup>th</sup> generation of the FLUKA project.

## Benefits:

- Extend user experience:
  - \* Give the **FLUKA community access to G4** features (geometry, scoring...) & physics models, directly from their existing simulation setups. *[Already at MOIRA stage]*
  - \* Give the **G4 community direct access to FLUKA physics** models (notably, hadronic models). *[Path to FLUKA++]*
- Ease and extend FLUKA vs G4 results inter-comparisons.
- Avoid duplications of efforts & codes where relevant (synergies with other codes).
- Ease addition of new developments (increased modularity + code re-use).
- Ease contributions in a collaborative environment.
- Ease maintenance.



# DETAILED ROADMAP TO FLUKA++

---

1) **Create the technical infrastructure (C++) [Moirai]:**

input/output through G4 UI commands, both FLUKA and G4 geometries support, source term, variance reduction, scoring, tracking tuning, magnetic fields, post processing...

2) In parallel, small **extensions needed in FLAIR to support both FLUKA and MOIRA inputs.**

3) **Interface with the full FLUKA hadronic model.**

Fortran FLUKA4 code, callable both from FLUKA4 and from MOIRA.

4) At this point, we can **provide a first public release, with name MOIRA (not as FLUKA).**

An application that can run G4 from both FLUKA and G4 input files.

5) **Modularize the present FLUKA physics components** (hadronic, electromagnetic, evaporation...).

Initially as Fortran modules, callable both from FLUKA4 and from MOIRA.

6) Interface to the FLUKA modules, as **new physics processes** in FLUKA++.

7) **Release to public FLUKA++**, when the FLUKA physics is fully integrated, and one gets equivalent or better performance than the present FLUKA code.

8) **Convert modules to C++, and improve performances.**

present

1 year

5 years?

# MOIRA: STRUCTURE & LINKS WITH G4

---

## Geometry & navigation

G4 geometry + zones 'a la FLUKA' + fully custom navigator.

## Materials

Directly from G4.

## Magnetic fields

Custom implementations directly **deriving** from G4.

## Cuts

Directly from G4.

## Run & stepping

Custom implementations **deriving** from G4 / directly from G4.

## Seeding

Directly from G4 + custom implementation **deriving** from G4, for event restart mode.

## Scoring

Fully custom.

## Biasing

Custom implementations directly **deriving** from G4.

# MOIRA: CURRENT STATUS

## Geometry & navigation

- Almost complete: support of FLUKA zones + lattice capabilities.
- To do: navigator needs extra debugging + voxels support.

## Materials

Extra treatment for *vacuum* and *blackhole*.

## Magnetic fields

Compatible with latest FLUKA implementation (multipoles, 2D/3D interpolated fields, user-defined).

## Cuts

To do: support production cuts expressed in energy.

## Run & stepping

Scoring quantities, differential variables, scorers, and run management.

## Seeding

Restart a simulation after crash (in a multi-threaded run), directly from the crashed event, with no extra I/O.

## Scoring

- 0D: per region
- 1D/2D: boundary crossing
- 1D: tracklength in a volume
- 3D: mesh scoring, in cartesian and cylindrical modes

Custom histogramming classes. Quantities can be cached and are shared among scorers.

## Biasing

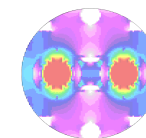
Custom implementations directly deriving from G4: region importance, weight windows, interaction length biasing.

## Flair

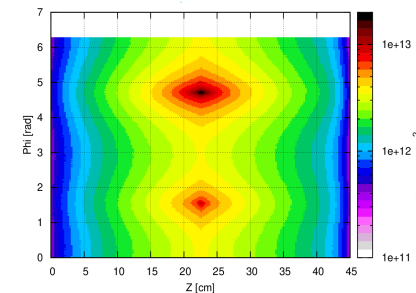
Input files conversion is operational.

## Testing

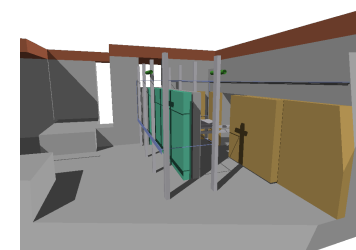
Few users are already using/testing MOIRA (CHARM, nTOF, LHC collimation).



MBRB



20 GeV proton gun in lead target  
Scoring fluence · cylindrical mesh



BLM benchmarks  
at the CHARM facility

# MOIRA: LATEST DEVELOPMENTS

---

## General scoring design

Addition of base scorer, quantities, differential variables, data collection, histogram classes.

## Volume identification scheme

Added fast volume identification scheme, supporting **replica / parametrized / division volumes**.

## Boundary crossing scoring + Double differential scoring

Support in MOIRA of the equivalent of the **USRBDX** and **USRYIELD** FLUKA cards.

## Fast mesh scoring

**Standalone, analytical implementation of the mesh scoring** (multi-threaded environment). Equivalent to entire G4 scoring manager and parallel world codebase. Support for both cartesian and cylindrical modes.

## Making FLUKA Physics models available to G4

Made it possible to call FLUKA hadron-nucleus interactions (cross-sections & final states) from any G4 run. Available as a PhysicsList.

- **Allow the community to access FLUKA Physics directly from the (G4-based) experiments frameworks** (CMSSW, Athena...).
- **Can be used by any G4-based application, including MOIRA.**

# MOIRA: LATEST DEVELOPMENTS

---

## Event-based seeding and simulation restart

Added ability to **restart a simulation after crash** (in a multi-threaded run), directly from the crashed event. Solution **without any extra I/O** (work on random number generator status).

## Radioactive decay modelling

Add possibility to **import irradiation profiles and cooling time**.

## Plug-ins for user routines

Compatibility with FLUKA user routines.

## User interface

Redesign of the scoring user interface (input file format, geometry-grouping logic, support for generic scorers).

## Flair

- Creation of **MOIRA cards** (progressive).
- **Inputs conversions: FLUKA ↔ MOIRA cards** (progressive).

# FAST MESH SCORING

---

# SUMMARY

---

## Motivation

- No loss of critical FLUKA functionalities while migrating to FLUKA++. Different philosophy / needs than G4.
- Mesh definition should **not interfere with tracking**. In G4, particle steps are dependent on the mesh bin size (parallel worlds are used to *define* the step length).
  - It is frequent for FLUKA users to define a very significant number of meshes for a study. Hence, possible runtime **performance and memory usage improvements** are critical.

## Proposal

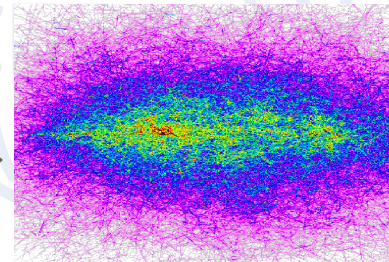
A **fully standalone, analytical implementation** of the mesh scoring.

- Equivalent to a replacement of the **entire G4 scoring manager and parallel worlds** implementations.
- Support for **both cartesian and cylindrical** modes.
- Includes all **necessary features**: user interface, geometry algorithms, mesh scoring data allocation and collection, handling of scoring data synchronization among threads...
- **Multi-threaded** environment.
- Standard mesh **transformations** (translations and rotations) are also supported.
- Results can be **dumped** in debug mode or in **binary** format (visualizable with Flair).

**Results:** Fully consistent with those obtained with the G4 implementation.

**Achieved run-time performance:** From  $\sim$  x40 to x100 **speed-up** versus G4, in the tested studies.

**Very significant reduction of the RAM usage:** generally **one order of magnitude**.



# IMPLEMENTATION DISCREPANCIES WITH G4

---

## Navigation in mesh geometry

- **G4:** Parallel world concept, relying on volumes replicas and custom navigator.
  - Can be computationally heavy (copying steps, dedicated replicas navigation).
  - Particle history modified when changing mesh binning size.
- **Moir:** Analytical geometry algorithms (do not modify the steps).

## Choice of scoring data containers

- **G4:** Maps. Implies that memory is allocated on the fly in the stepping action. Allows the data to be sorted on the fly, per bin.
- **Moir:** Vectors with pre-allocated memory (at run construction, for each master and worker thread): scores are only *assigned* during the stepping action.

## Scoring data size limit & synchronization with master (worker threads)

- **G4:** Size of the entire mesh, for each worker thread. Worker container is flushed at the end of each event.
- **Moir:** Size limit per worker thread. Call synchronization construct whenever worker container is full (within an event), or in any case, at the end of the run.  
Drawback: Does not allow for the statistics on mesh scores to be event-based, like in G4. Statistics hence must rely on a *run-based* granularity (post-processing script, consistent with current FLUKA approach).

# MESH GEOMETRY NAVIGATION ALGORITHMS

## Description

Split any step along mesh bins: returns all encountered path lengths and bins indices.  
Replace the parallel worlds in G4.

Minimal number of arithmetic operations:

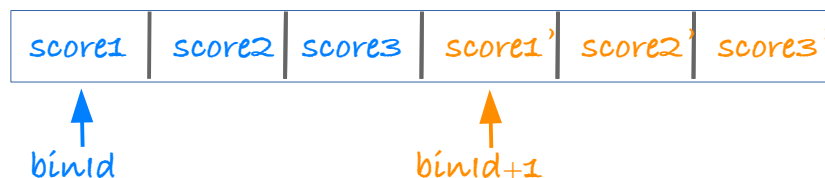
- Cartesian mode: can be performed with **only additions / subtractions (no division!)**.
- Cylindrical mode: can be performed with **no tan / arctan**.

## Key ideas

- Quickly discard steps not crossing the mesh, **without doing any projection** (use **pre-computed envelope**).
- When step is not discarded, and step point(s) are **outside of mesh**: project them to the mesh.
- Calculate, **before step splitting**, the **total step length** through the mesh.  
Handle very frequent corner case: **departure and destination points are in the same mesh bin** (no norm calculation!).
- **`Snake` algorithm**: Keep **running (cached)** path length, distance to the next closest bin, and bin indices.
- Phi bin borders tans are cached at initialization. For each step, make a binary search in that cache to find point's phi.
- **Computations only done in relevant dimensions**: dimensions to work on are progressively decreased, while the **`running` point** progressively moves towards destination.

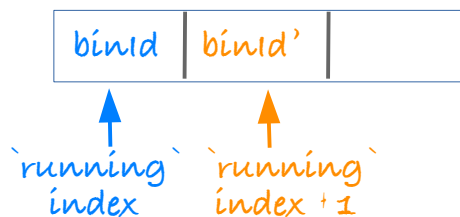
# MESH SCORING DATA CONTAINERS

- All data collectors are pre-allocated at run construction (in the master thread and in each worker thread).
- Vectors are filled on the fly in the stepping action.
- MeshMasterData collector: Container size = Number of bins in the mesh.

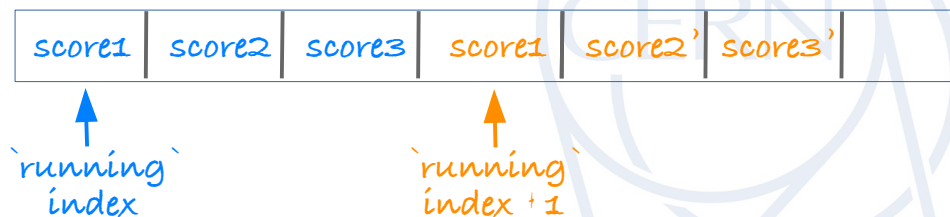


- MeshWorkerData collector: Container size = Limited by hardcoded constant.

Bins vector:



Scores vector:



# MESH SCORING STEPPING

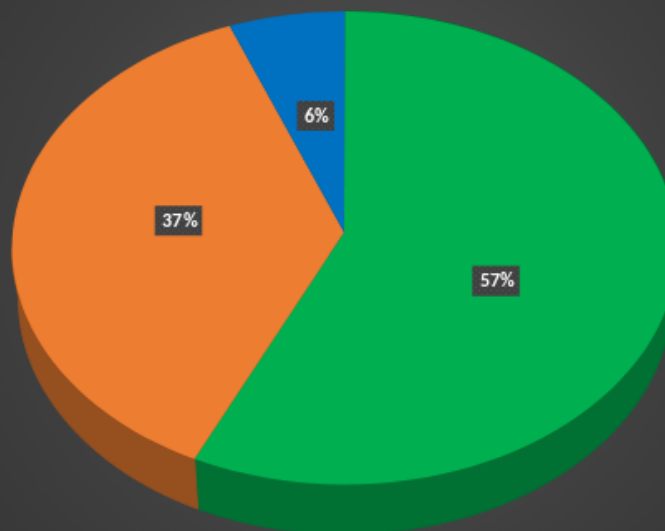
---

For each step:

- Checks that the step crosses the mesh **envelope**.
- If so, checks that the step actually **crosses the mesh**.
- If so, check that any scorer associated to the mesh passes the **filtering**.
- If so, **compute the score** (only for scorers passing filtering).
- If *any* score is non null: **call the mesh step splitting algorithm**.
- **For each crossed bin:**
  - If worker data collector is full, **flush its data** to master thread, and **reset** worker data container *running index*.
  - In the **case where the present bin was already encountered (last)**: do not increase *running index*. Instead, stack up scores associated with the same bin.
  - **Otherwise, increment** the worker *running index*.  
**Append** present binIndex and score to the worker data collectors (bins and scores vector).

# MESH SCORING RUNTIME

MESH SCORING RUNTIME



- GEO ALGORITHMS
- WRITE SCORING DATA TO WORKER CONTAINER
- LOCKING MUTEX + FLUSH DATA INTO MASTER

Scoring  
total energy  
deposition  
and fluence.

FTFP\_BERT\_HP  
LIV.

20 MeV  
neutron gun.

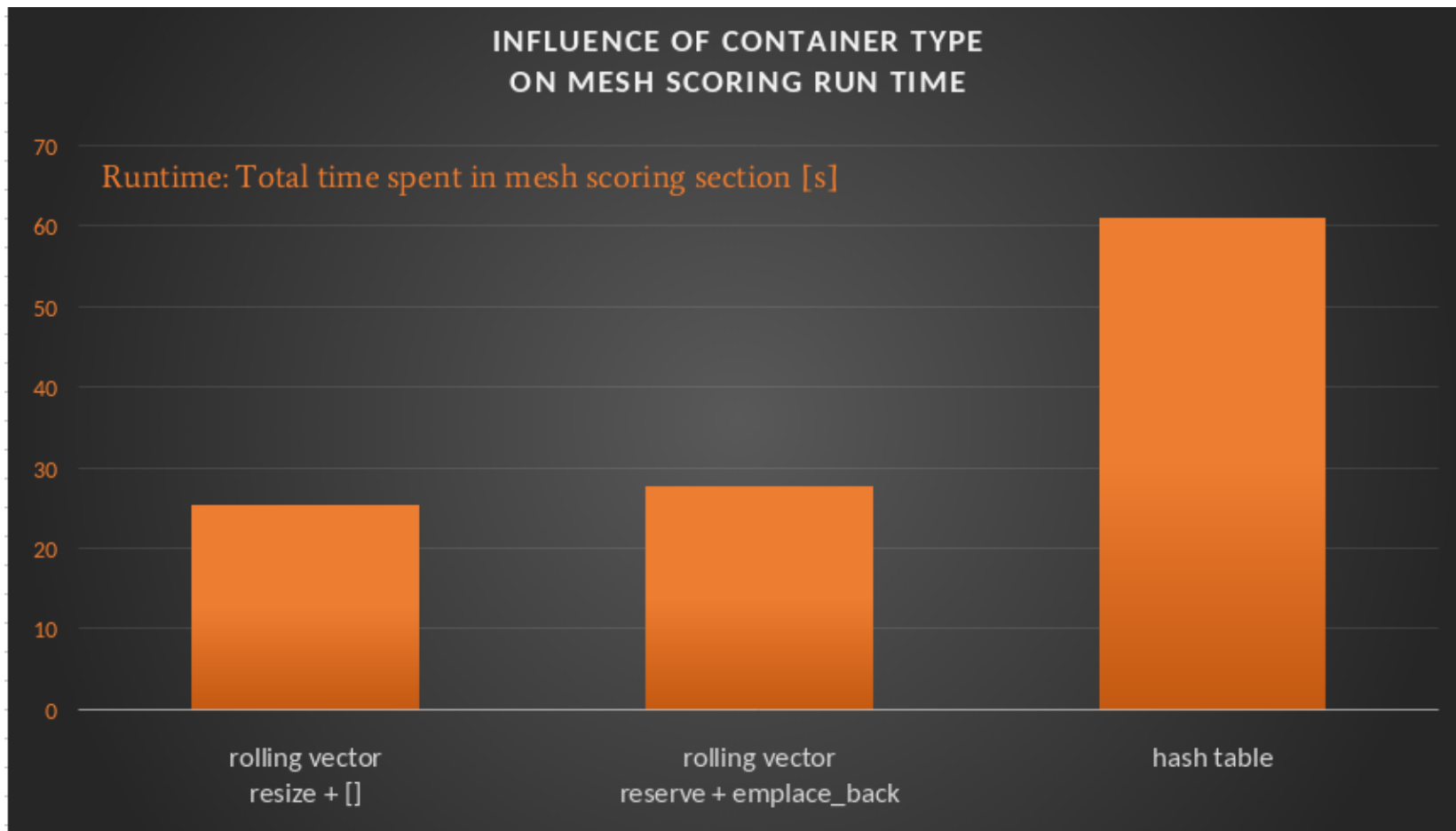
Cartesian  
mesh,  
**Air + Conc.**,  
60x60x45 cm,  
with  
240x240x180  
bins.

100 000  
events.

## All benchmarks performed on:

CPU: Intel(R)  
Core(TM) i7-7820HQ  
CPU @ 2.90GHz  
Architecture: x86\_64  
# CPUs: 8  
# Threads per core: 2  
Cache: L1 32 KB, L2  
256 KB, L3 8 MB  
RAM: 32 GB  
Operating system:  
Cern CentOS 7

# OPTIMIZATION: CHOICE OF WORKER DATA CONTAINER



Scoring  
total energy  
deposition  
and fluence.

FTFP\_BERT\_HP  
LIV.

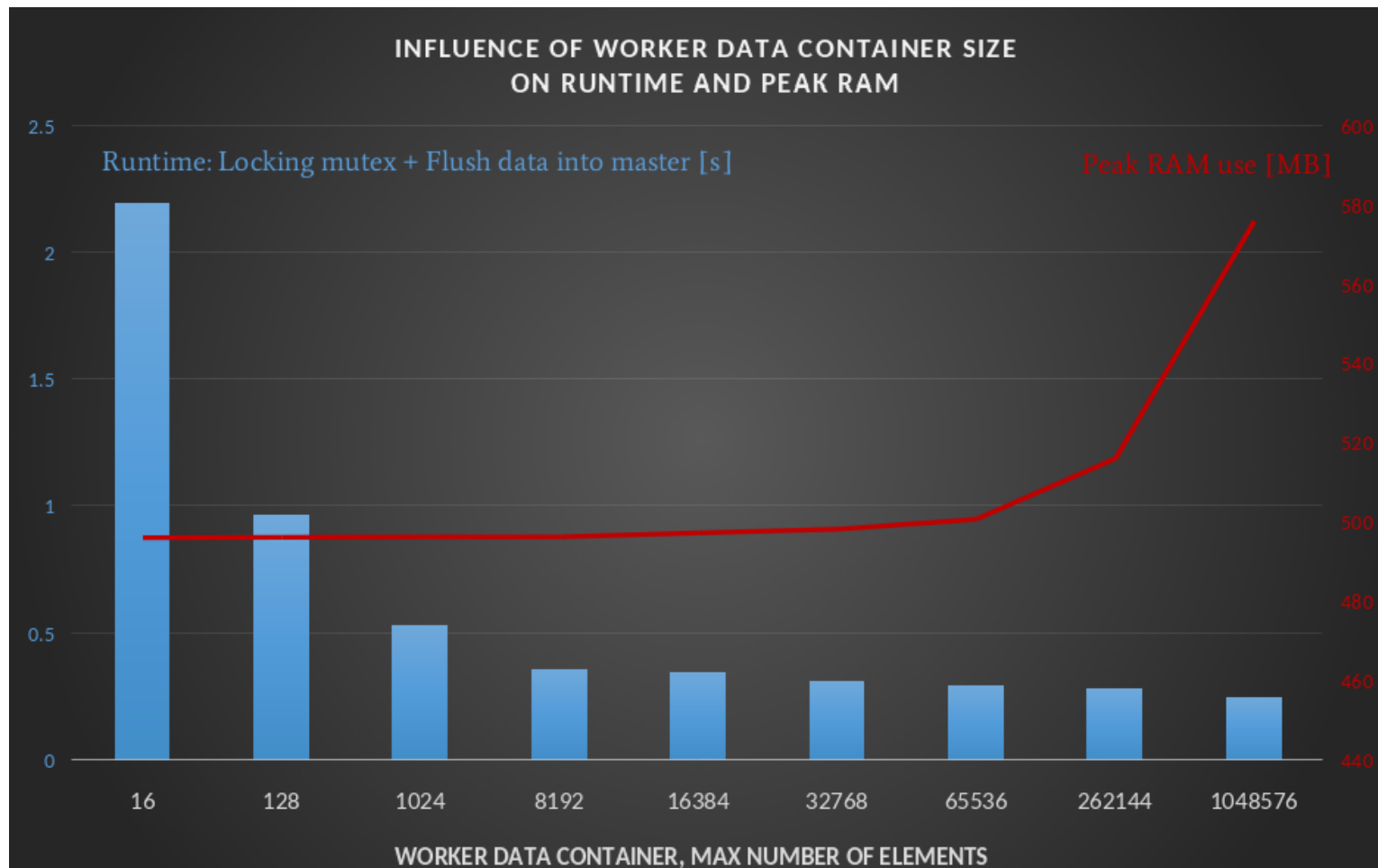
20 MeV  
neutron gun.

Cartesian  
mesh,  
**Air + Conc.**,  
60x60x45 cm,  
with  
240x240x180  
bins.

10 000 000  
events.

- Clear benefits of the rolling vector implementation.

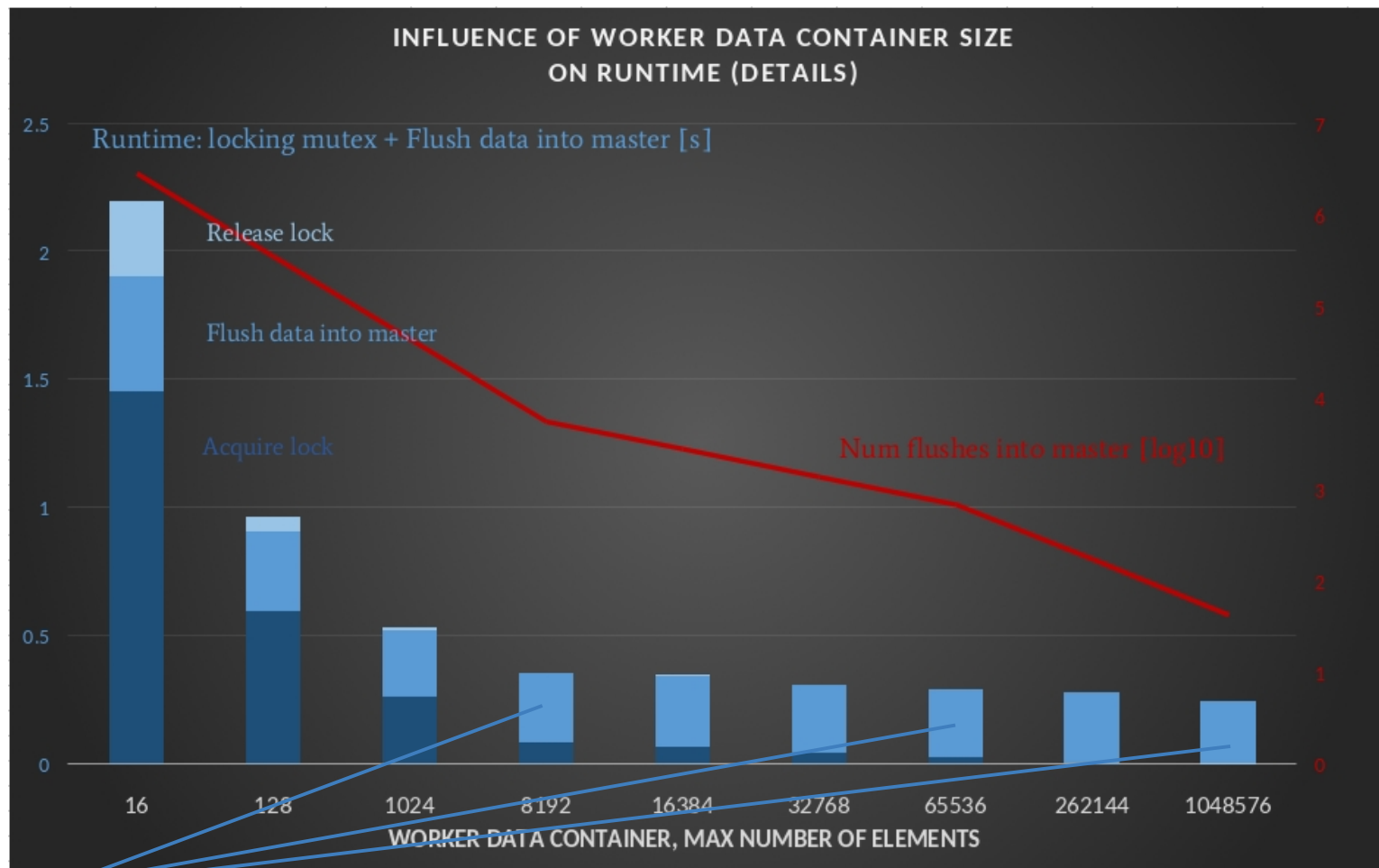
# OPTIMIZATION: CHOICE OF WORKER DATA CONTAINER SIZE



Scoring  
total energy  
deposition  
and fluence.  
FTFP\_BERT\_HP  
LIV.  
20 MeV  
neutron gun.  
Cartesian  
mesh,  
**Air + Conc.**,  
60x60x45 cm,  
with  
240x240x180  
bins.  
1 000 000  
events.

- Chose  $2^{15}$  elements (32 768) as max number of elements in worker data container.

# MESH SCORING OPTIMIZATION: CHOICE OF WORKER DATA CONTAINER SIZE



Scoring  
total energy  
deposition  
and fluence.

FTFP\_BERT\_HP  
LIV.

20 MeV  
neutron gun.

Cartesian  
mesh,  
**Air + Conc.**,  
60x60x45 cm,  
with  
240x240x180  
bins.

1 000 000  
events.

- With a relevant data container size, most of the data transfer run time is spent in data transfer (and not in mutex locking!).

# RUN RESULTS: G4

Scoring fluence.

FTFP\_BERT\_HP LIV.

20 GeV proton gun.

Cartesian mesh.

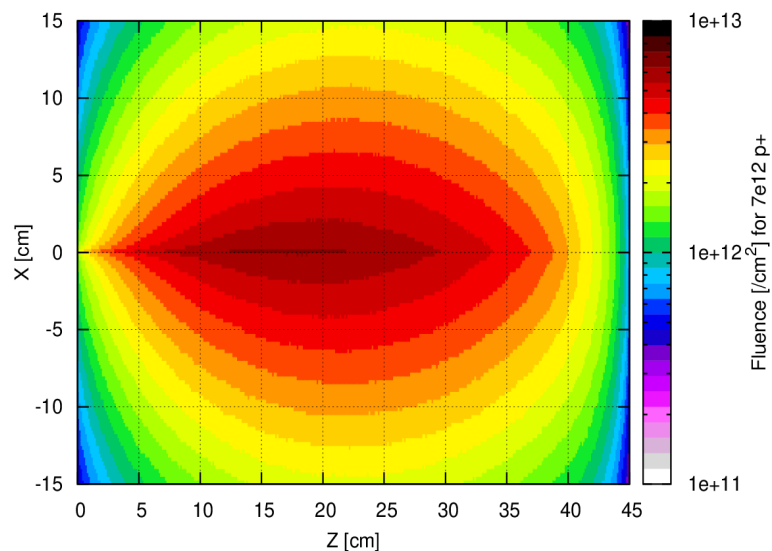
Lead.

60x60x45 cm.

with 240x240x180 bins.

40 000 events.

Fluence G4



Scoring total energy deposition.

FTFP\_BERT\_HP LIV.

20 GeV proton gun.

Cartesian mesh.

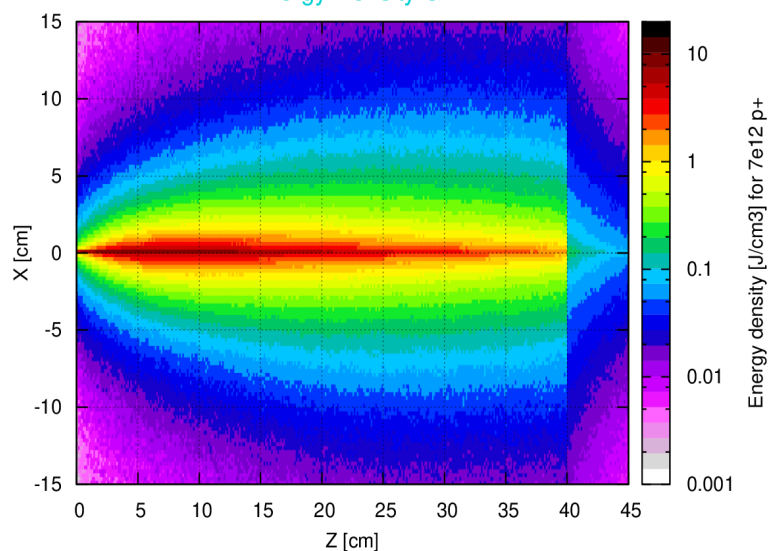
Lead.

60x60x45 cm.

with 240x240x180 bins.

40 000 events.

Energy Density G4



# RUN RESULTS: CUSTOM IMPLEMENTATION

Scoring fluence.

FTFP\_BERT\_HP LIV.

20 GeV proton gun.

Cartesian mesh.

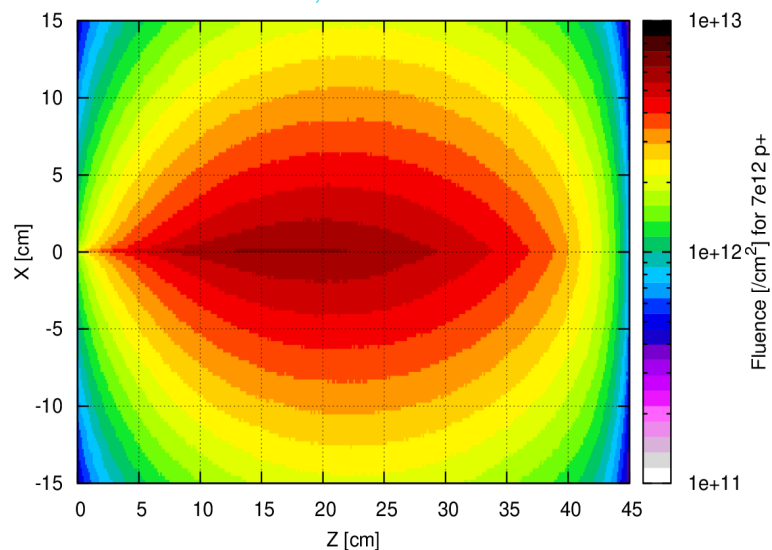
Lead.

60x60x45 cm.

with 240x240x180 bins.

40 000 events.

Fluence CUSTOM, NO PARALLEL WORLD



Scoring total energy deposition.

FTFP\_BERT\_HP LIV.

20 GeV proton gun.

Cartesian mesh.

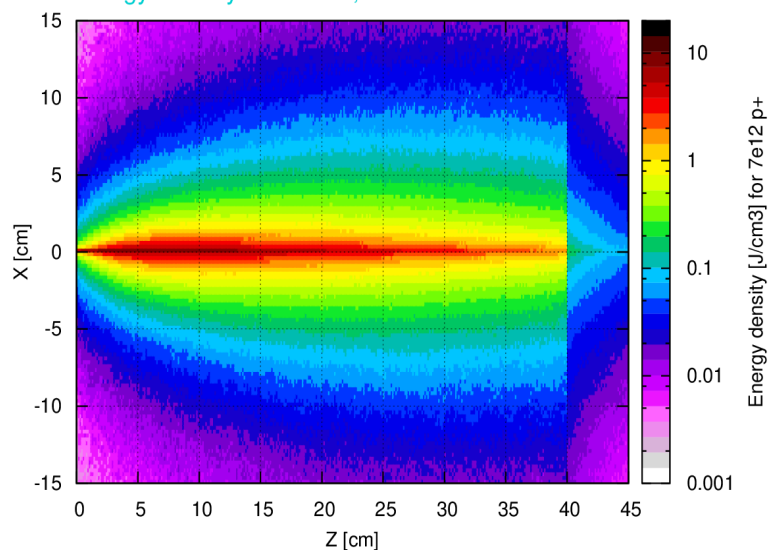
Lead.

60x60x45 cm.

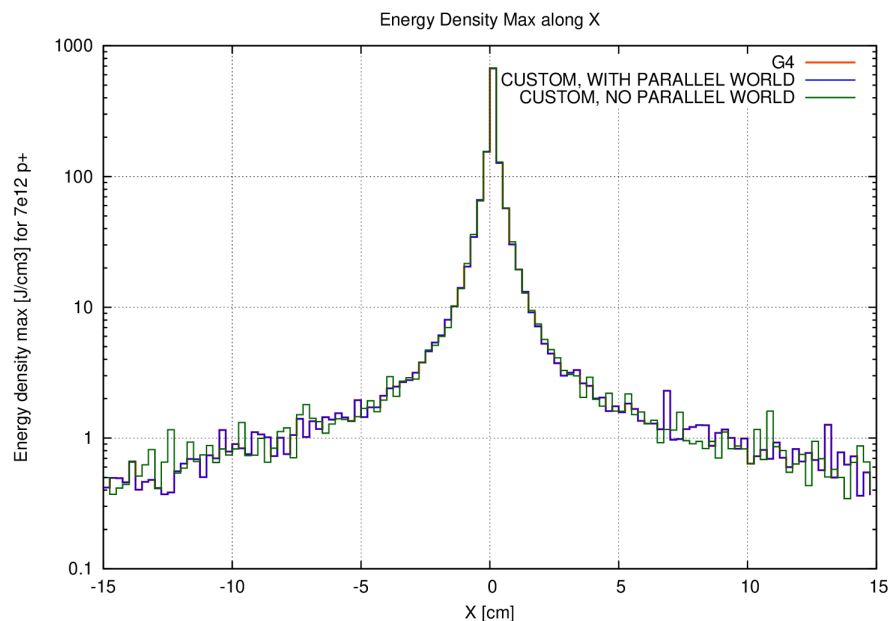
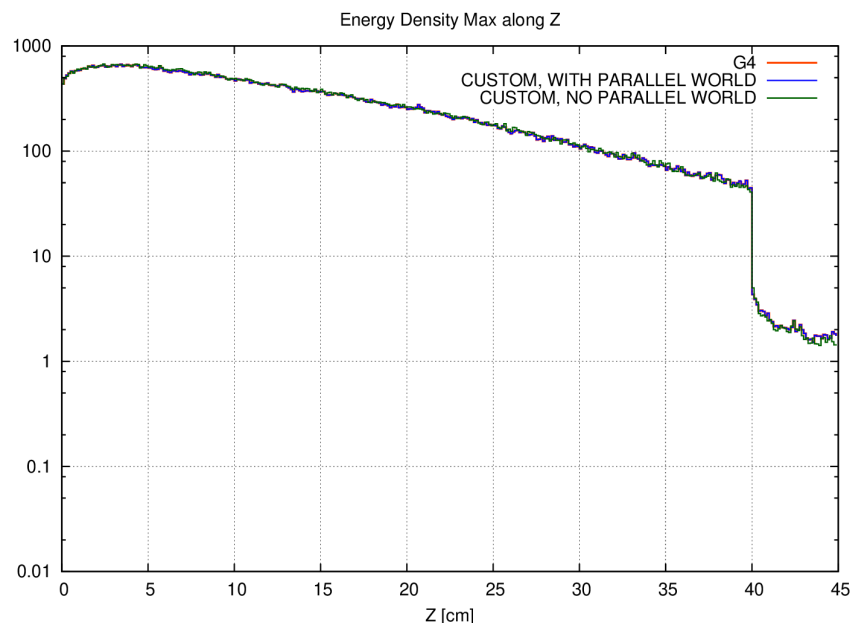
with 240x240x180 bins.

40 000 events.

Energy Density CUSTOM, NO PARALLEL WORLD



# RUN RESULTS: CUSTOM IMPLEMENTATION VERSUS G4



Scoring  
total energy  
deposition.

FTFP\_BERT\_HP  
LIV.

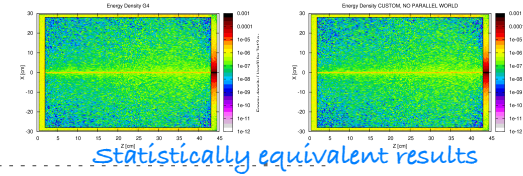
20 GeV  
proton gun.

Cartesian  
mesh,  
Lead,  
60x60x45 cm,  
with  
240x240x180  
bins.

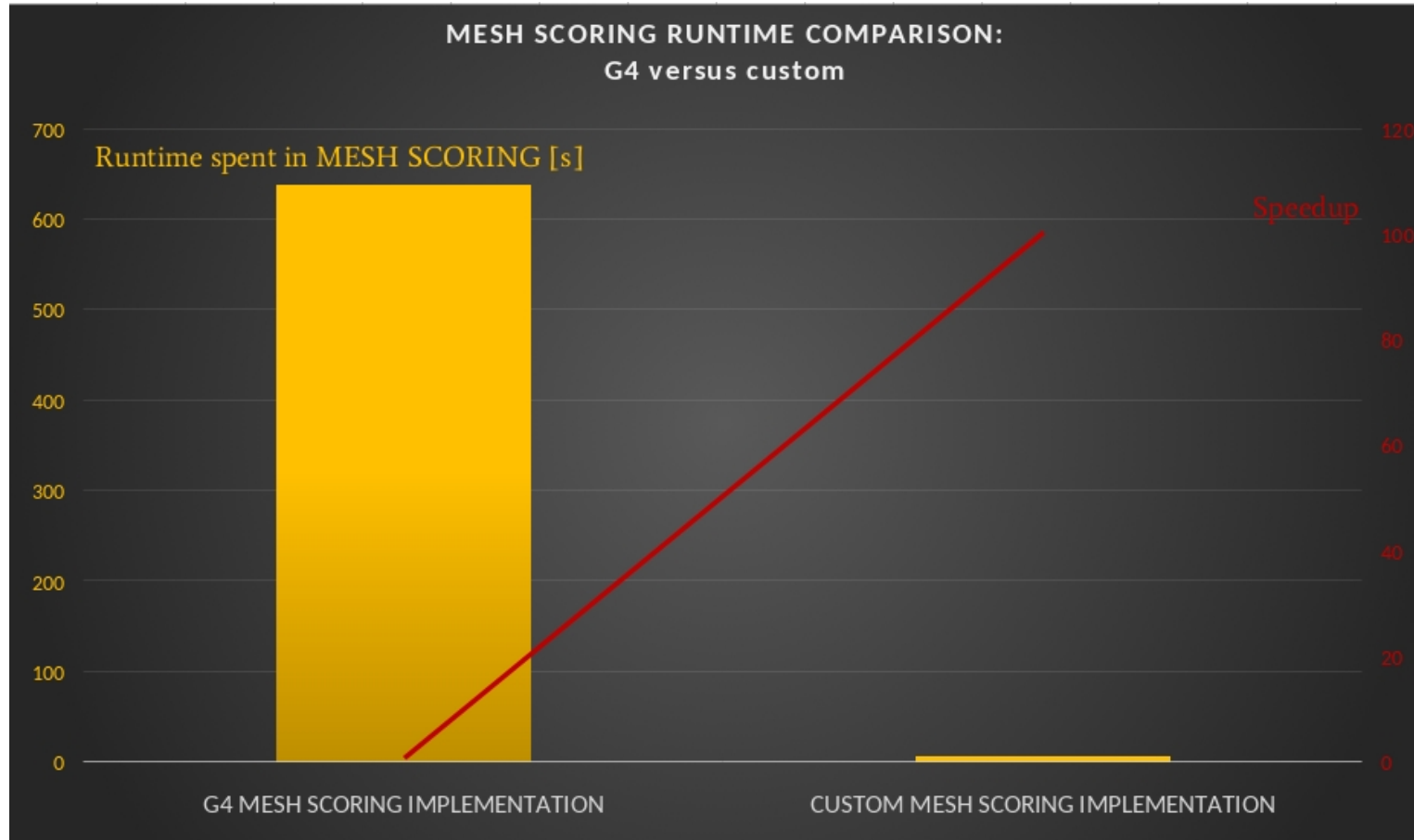
40 000  
events.

- Activating the G4 parallel worlds in the custom implementation: results binwise identical with G4.
- Deactivating the G4 parallel worlds in the custom implementation: results statistically equivalent to G4.

# PERFORMANCE VS G4 (MESH SCORING)



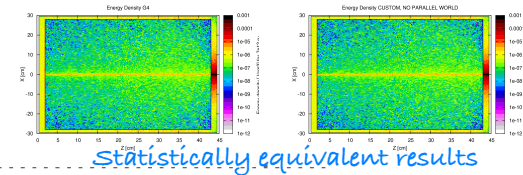
Statistically equivalent results



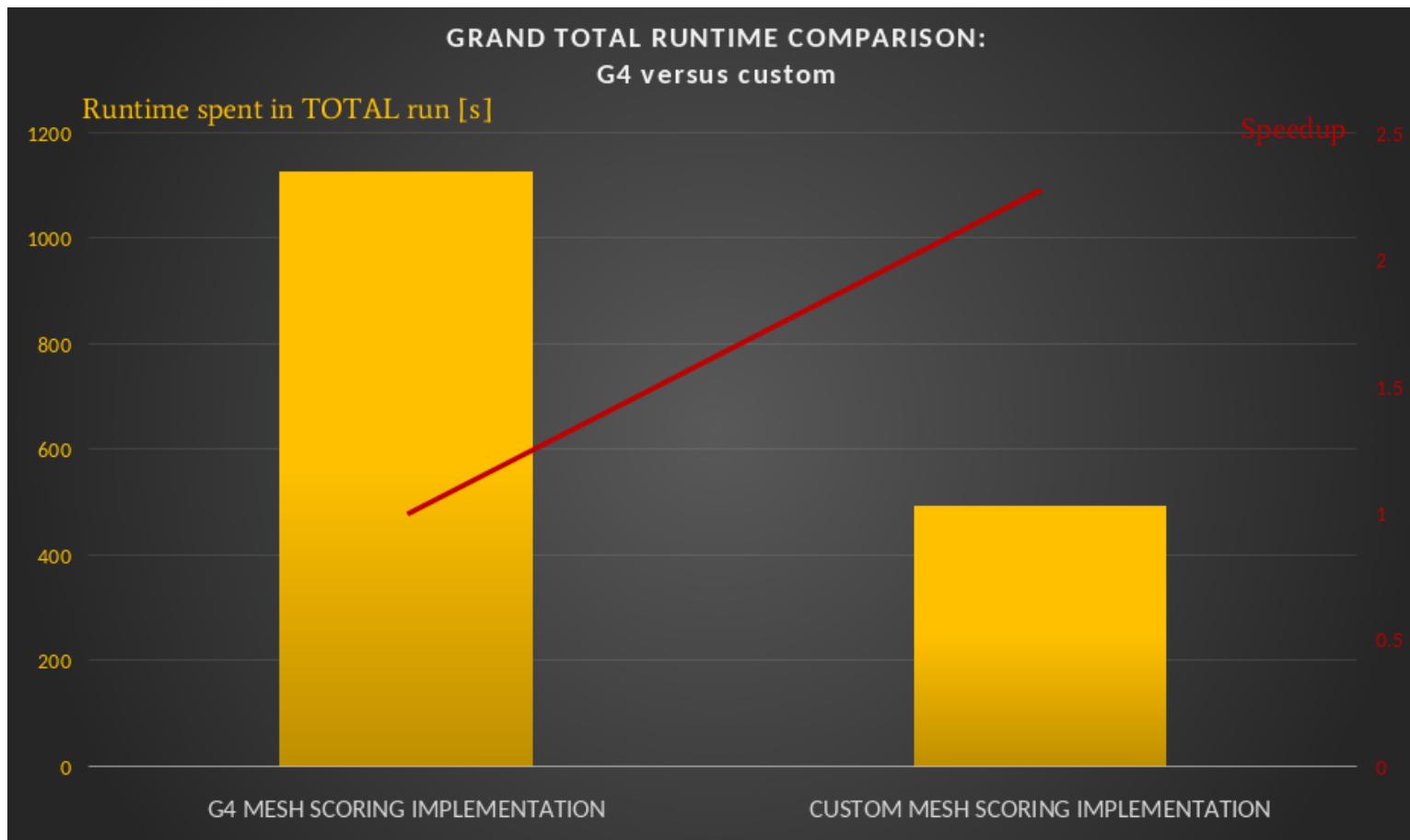
Scoring  
total energy  
deposition  
and fluence.  
  
FTFP\_BERT\_HP  
LIV.  
  
20 MeV  
neutron gun.  
  
Cartesian  
mesh,  
Air + Conc.,  
60x60x45 cm,  
with  
240x240x180  
bins.  
  
5 000 000  
events.

- ~ x100 speedup in TOTAL MESH SCORING RUN TIME (geo step splitting, saving scoring data, flushes to master...).

# PERFORMANCE VS G4 (TOTAL RUN TIME)



Statistically equivalent results



Scoring  
total energy  
deposition  
and fluence.

FTFP\_BERT\_HP  
LIV.

20 MeV  
neutron gun.

Cartesian  
mesh,  
Air + Conc.,  
60x60x45 cm,  
with  
240x240x180  
bins.

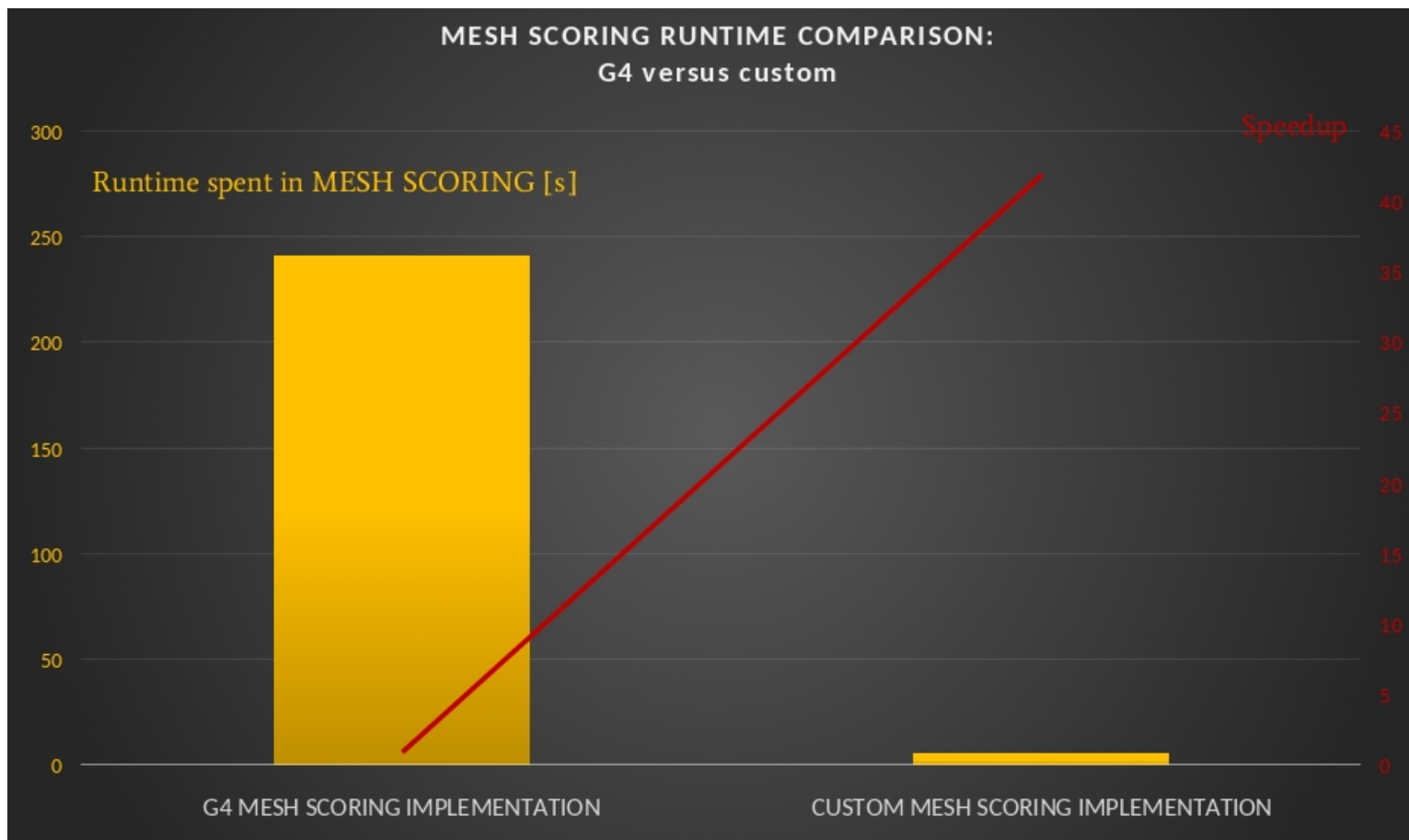
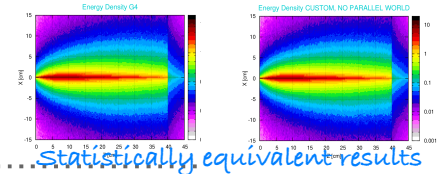
5 000 000  
events.

Peak RAM use:

- G4:  
5.6 GB
- CUSTOM  
implementation:  
515 MB

- ~ x2.3 speedup in TOTAL RUN TIME (entire moira run, including all initialization, geometry construction...).

# MESH SCORING: G4 VS CUSTOM IMPLEMENTATIONS



Scoring  
total energy  
deposition  
and fluence.

FTFP\_BERT\_HP  
LIV.

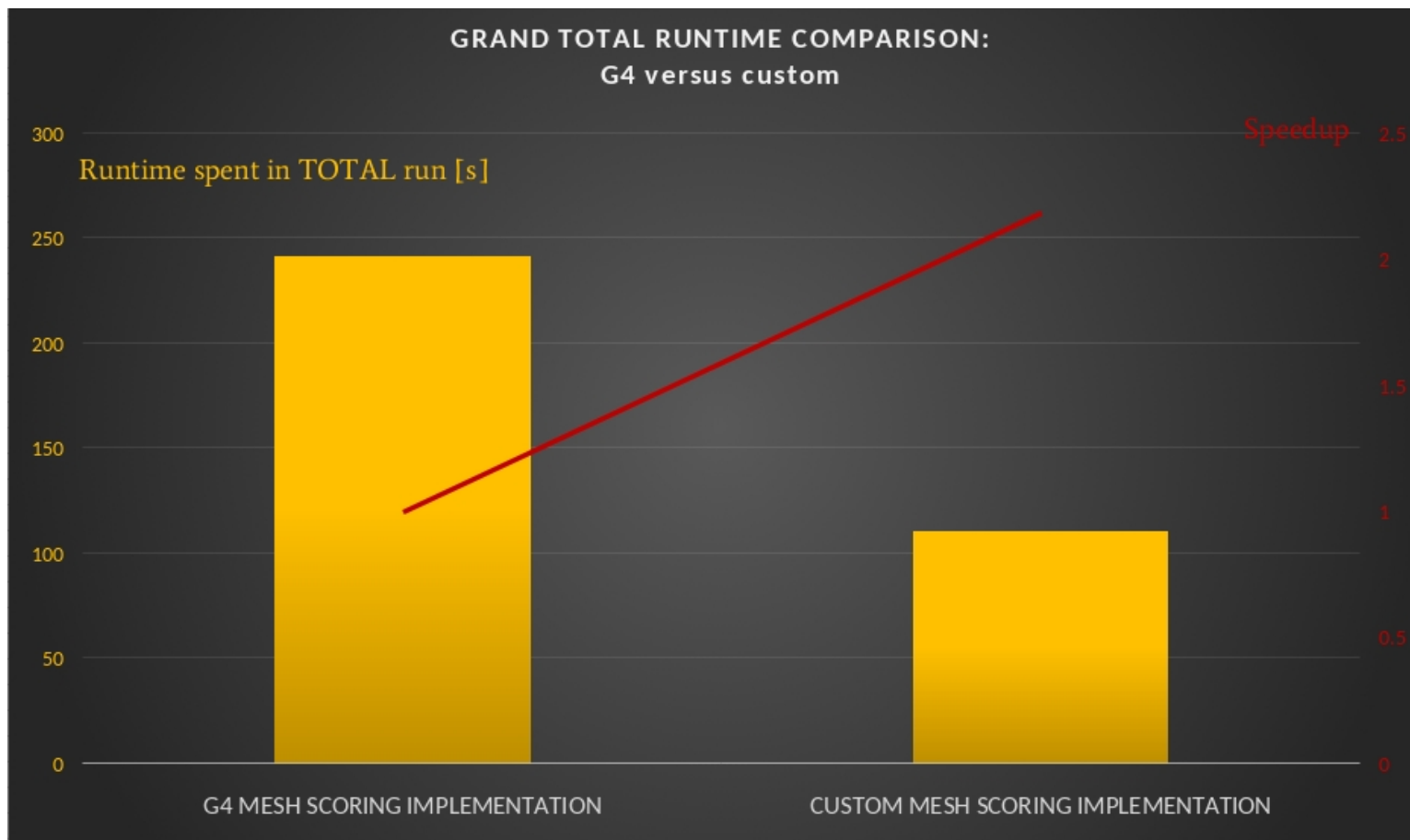
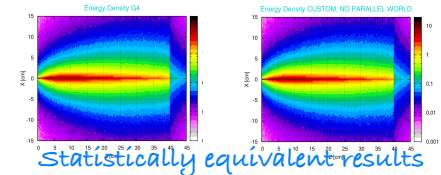
20 GeV  
proton gun.

Cartesian  
mesh,  
Lead,  
60x60x45 cm,  
with  
240x240x180  
bins.

400 events.

- ~ x42 speedup in TOTAL MESH SCORING RUN TIME (geo step splitting, saving scoring data, flushes to master...).

# PERFORMANCE VS G4 (TOTAL RUN TIME)



Scoring  
total energy  
deposition  
and fluence.

FTFP\_BERT\_HP  
LIV.

20 GeV  
proton gun.

Cartesian  
mesh,  
Lead,  
60x60x45 cm,  
with  
240x240x180  
bins.

400 events.

Peak RAM use:

- G4:  
6.4 GB
- CUSTOM  
implementation:  
519 MB

- ~ x2.2 speedup in TOTAL RUN TIME (entire moira run, including all initialization, geometry construction...).

# ANNEX

---

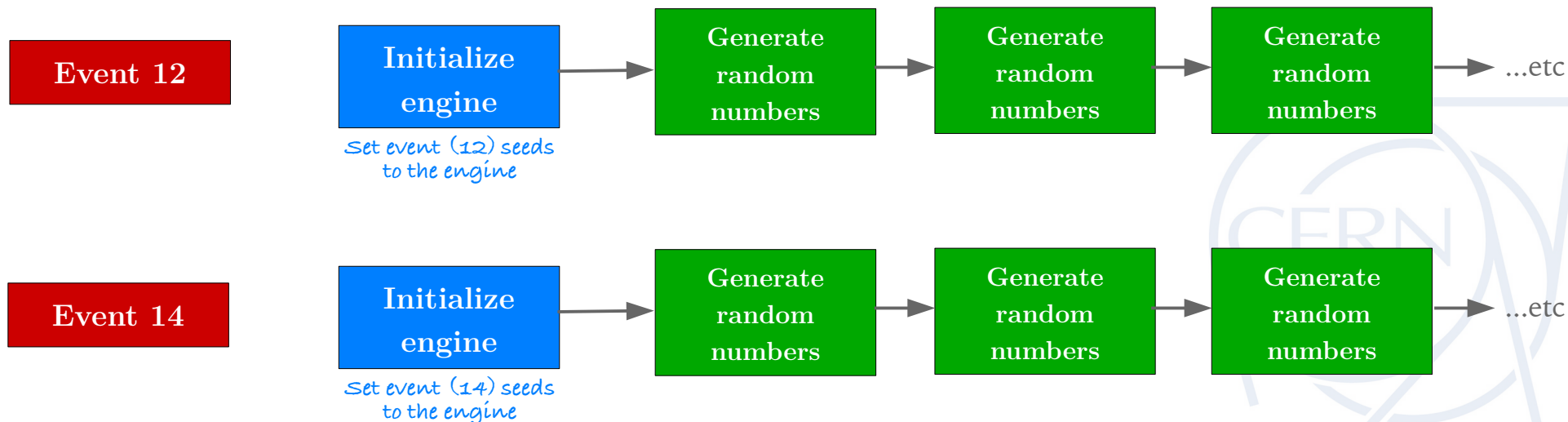
# SEEDING IN G4

Each event is associated with a given seed (actually, a pair of seeds).

- **At beginning of event processing:**  
the random engine to be used for the event, is initialized with this seed.
- **During event processing:**  
the random engine generates on the fly random numbers needed for the event.

**A given event id is always associated with the same seed, for any simulation.**

This guarantees **strong reproducibility** of the simulation results.



# SEEDING IN G4

---

**G4RNGHelper:** Seeds helper and storage.

Seeds are generated in **groups**.

They are stored in a **queue** in a G4RNGHelper instance.

The **group size** depends on the number of events to simulate.

If too many events: a **max number of seeds is generated** and stored in the queue (20 000 in G4).

**Re-seeding** will occur later on within the event loop, when relevant.

**Random engines** used in G4.

- Base class is **HepRandomEngine**.

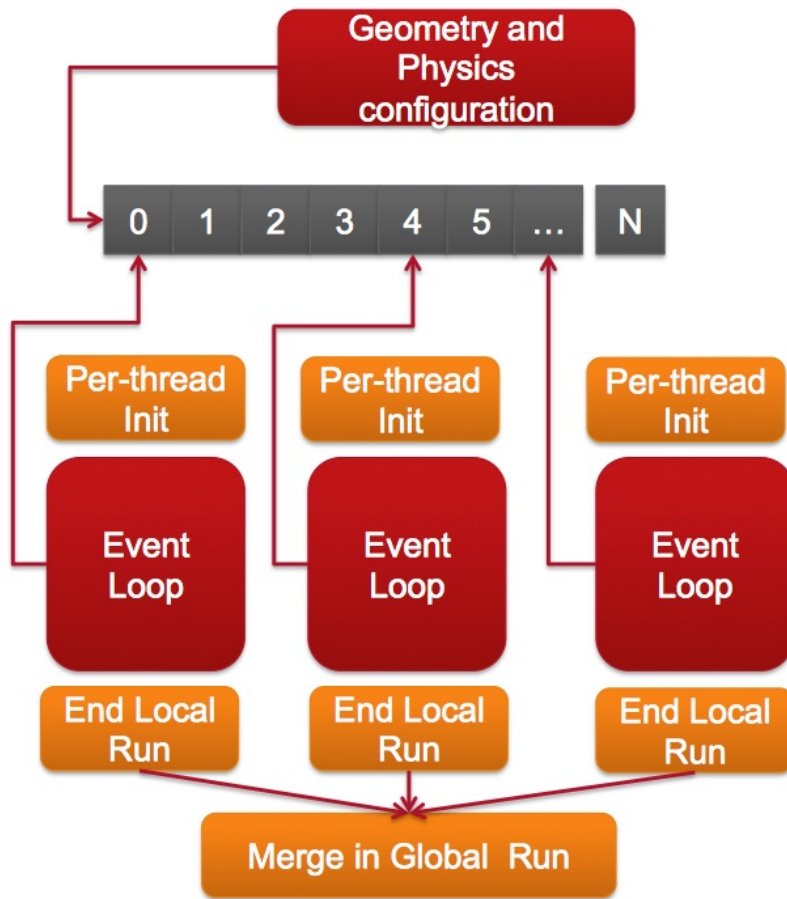
- Large choice of engines. Default is **MixMaxRng**.

With this engine, streams created from seeds differing by at least one bit, are guaranteed to be independent and non-colliding for at least the next  $10^{100}$  random numbers.

- Interface to access the engine through a wrapper class: **HepRandom**.

*HepRandom::flatArray, HepRandom::SetTheSeeds, HepRandom::saveFullState, etc.*

# SEEDING IN A G4 MULTI-THREADED RUN



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

Threads compete for next event to be processed

Command line scoring and G4tools automatically merge results from threads

**What will never be reproducible:**

- which thread get which events.
- exact number of events processed by a thread.

**What makes the simulation reproducible:**

Each event id is associated with a seed, **independently** of which thread actually processes the event!

From <https://indico.cern.ch/event/781244/contributions/3251900/attachments/1782717/2901032/Multithreading.pdf>

# SEEDING & SIMULATION RESTART: ISSUE AND PREVIOUS APPROACHES

---

## MOTIVATION

Be able to re-run a specific event, or sets of events, instead of re-launching a full run.

- Useful in case of crash, to avoid user waiting time!
- Not possible until now, in a multi-threaded run.

## STANDARD SOLUTION

Issue raised in 2019 to Geant4: [link](#)

For each event, dump the seed being used to an external file. Then, after a crash, just **re-use** the seed file of the crashed event.

- Internally in Geant4, there is a **simple call to reset the random engine status to the one of the seed file**.
- This approach works, but **performance issues**. The I/O for each event, depending on the simulation, can be very significant with respect to the total runtime spent in the event.

## OTHER IDEA (VARIANT): Dump events seeds into files AT REGULAR TIME INTERVALS

- Still requires extra I/O though.
- Actually, **cannot work in MT mode** (need to *compute* the event seed from the *run* seed).

# SEEDING & SIMULATION RESTART: SOLUTION

---

## NO I/O, ADVANCE MASTER THREAD RANDOM ENGINE STATUS

- Since **seeds are associated to event ids**, it is actually not even needed to **dump any seed file** to restart a specific event.
- Play directly with the random engine, by **advancing its status (master thread)**, then sending the computed **seed** to the worker thread. **Make as if event 0 (in the new run) is even N (crashed event)**.

## Solution design

- A **multi-threaded run manager** is added to MOIRA (deriving from G4).
- It has a **seeds initialization function**. **Reproduces the G4 master thread seeding**, up to the event one wants to **simulate**. **Re-seeding** is handled there as well, as needed.

## Additional benefits

- One can re-run in multi-threaded mode.
- **One can restart** not only the crashed event, but **a full simulation from the crashed event**.

This is particularly useful, as the event id of a specific crashed event might not be precisely known.

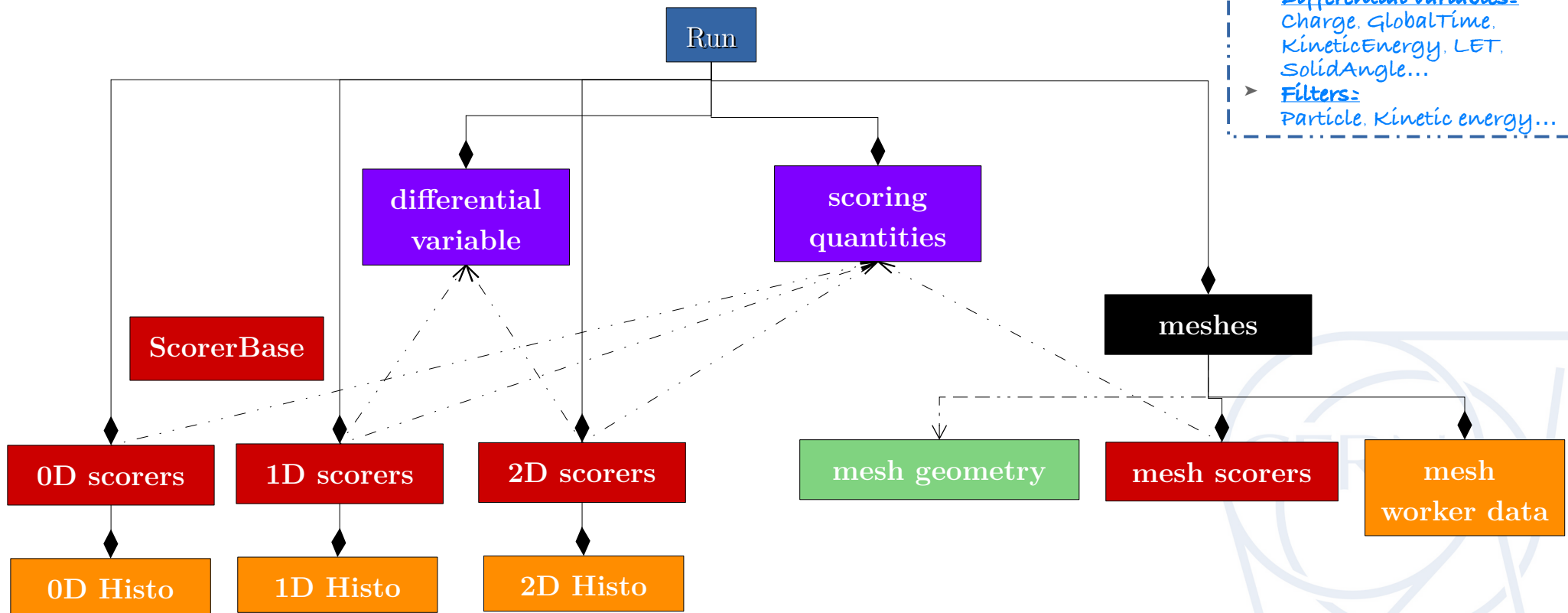
## How to use

- 1) Set the number of events to be re-launched in MOIRA macro, as usual.
- 2) This `reproduce a crashed event` mode is **inactive by default!**

Run: `moira input.m --runSeed crashed_run/run.rndm --startEventId myId`

# GENERAL SCORING OWNERSHIP DESIGN

IN EACH  
WORKER THREAD



- Scoring quantities:  
EMEnergy, Energy, NIEL,  
ParticleCount, Secondaries,  
TrackLength...
- Differential variables:  
Charge, GlobalTime,  
KineticEnergy, LET,  
SolidAngle...
- Filters:  
Particle, Kinetic energy...

# LATEST CORE DEVELOPMENTS: SCORING RUNTIME

## Master thread

## Worker thread

## Worker thread

- Construct master G4RunManager.
  - Leads to in chain construction of all managers (and their messengers): run, event, track, stacking, stepping...
- Detector + physics list + user actions are constructed and assigned to the master RunManager.
- Initialize seeds.
- Construct **geometry**.

CreateAndStartWorkers().

G4MTRunManager::MergeRun

Master run contain scoring data for each relevant scorer.  
Can do stats, dump data to files, draw, etc.

## 0D/1D/2D scoring

- G4UserWorkerThreadInitialization::CreateAndStartWorker
- G4WorkerRunManager and worker runs are constructed.
- Construct **meshes** and **scorers**, which will hold worker data.

Allocate memory for worker scoring data.

Fills (assigns!) scores to worker buffer.

No synchronization related to data filling!

Flush worker **EVENT** data to worker **RUN** data container.  
Reset worker **EVENT** data container.

MergeRun()

Call **synchronization** construct and **flush** **WORKER RUN** data to **MASTER** thread.

## Mesh scoring

- G4UserWorkerThreadInitialization::CreateAndStartWorker
- G4WorkerRunManager and worker runs are constructed.
- Construct **meshes** and **scorers**, which will hold worker data.

Allocate memory for worker scoring data.

Fills (assigns!) scores to worker buffer.

If worker buffer is **full** (for any of the meshes):  
Call **synchronization** construct and flush data to **MASTER** thread (for that mesh).  
**Reset buffer index, re-use** existing buffer.

MergeRun()

Call **synchronization** construct and **flush** remaining **WORKER** data to **MASTER** thread.

Master initialization

Workers initialization

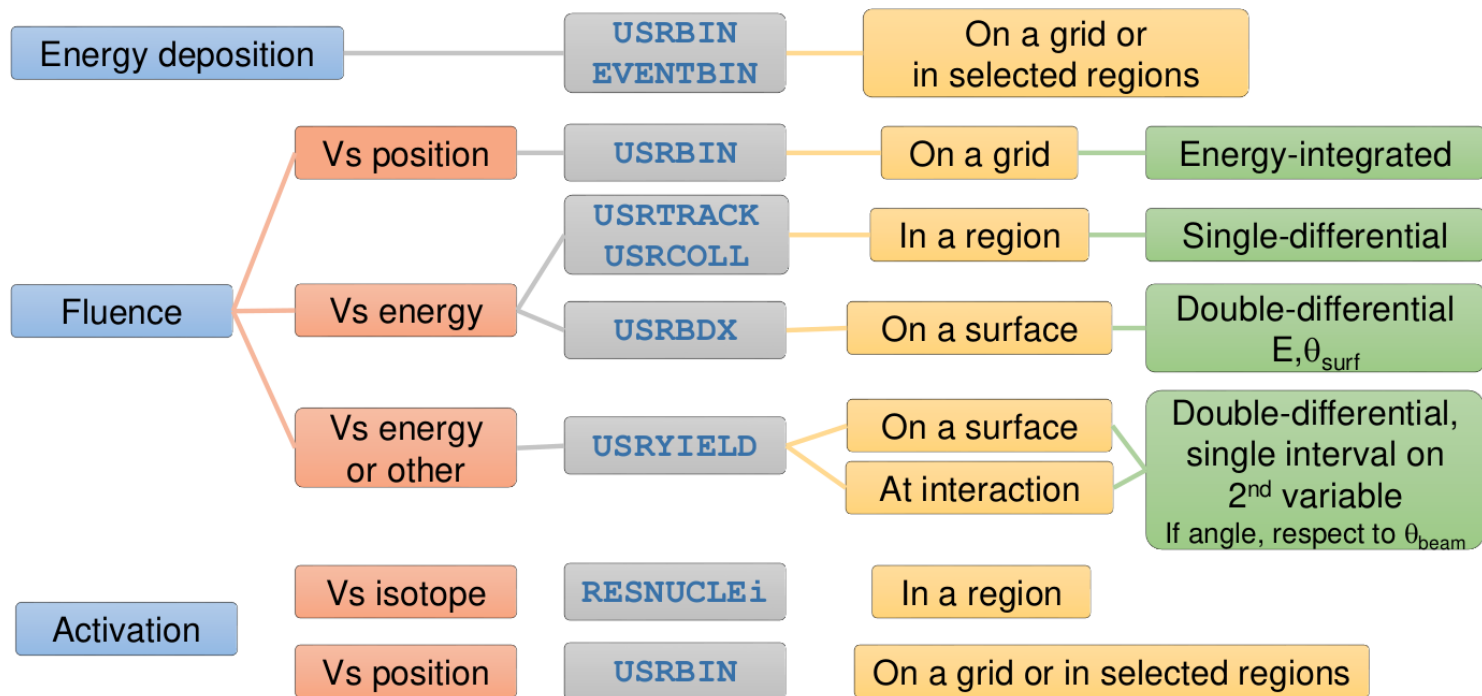
In event loop

End of event

End of run

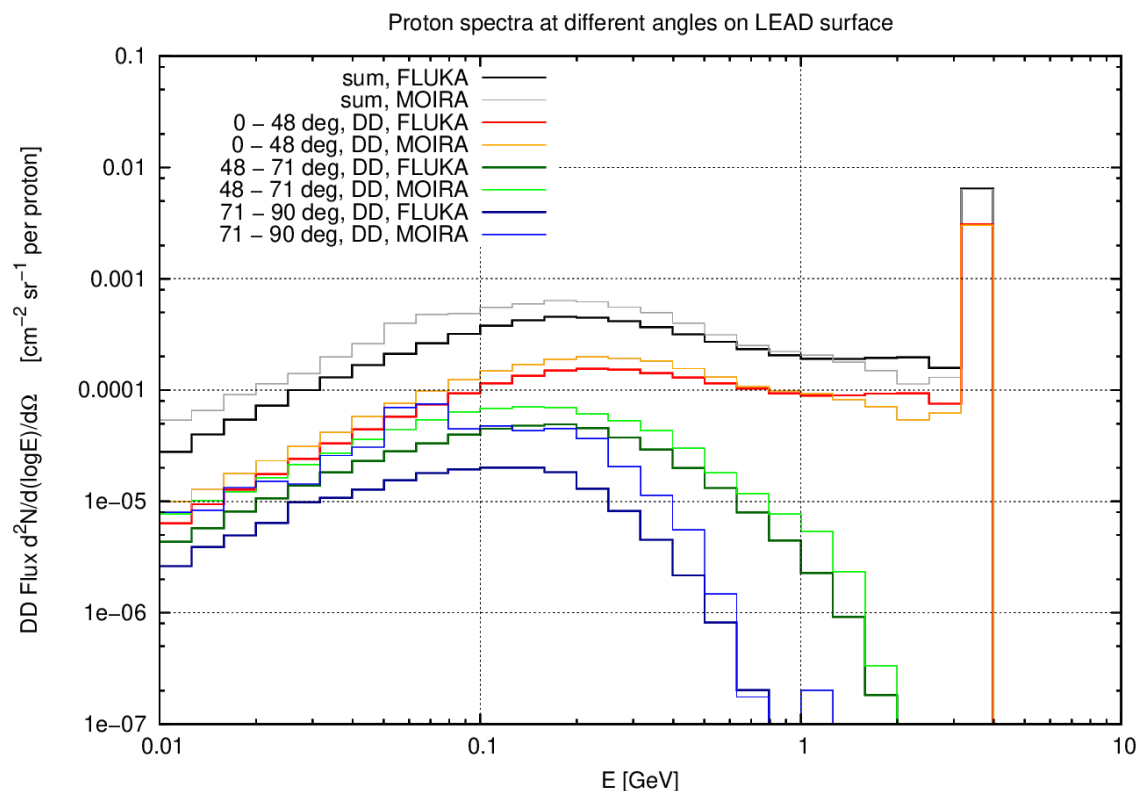
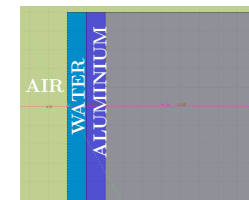
Post processing in master

## FLUKA estimator zoo



# LATEST CORE DEVELOPMENTS: DOUBLE DIFFERENTIAL SCORING

- Support in MOIRA of equivalent of USRBDX & USRYIELD.
- Thread-local **2D data collectors**, implemented in efficient way.
- Linearized 2D indices.
- Added relevant **differential variables**: Solid angle, LET...



Scoring proton flux on lead layer surface.

3.5 GeV proton gun.

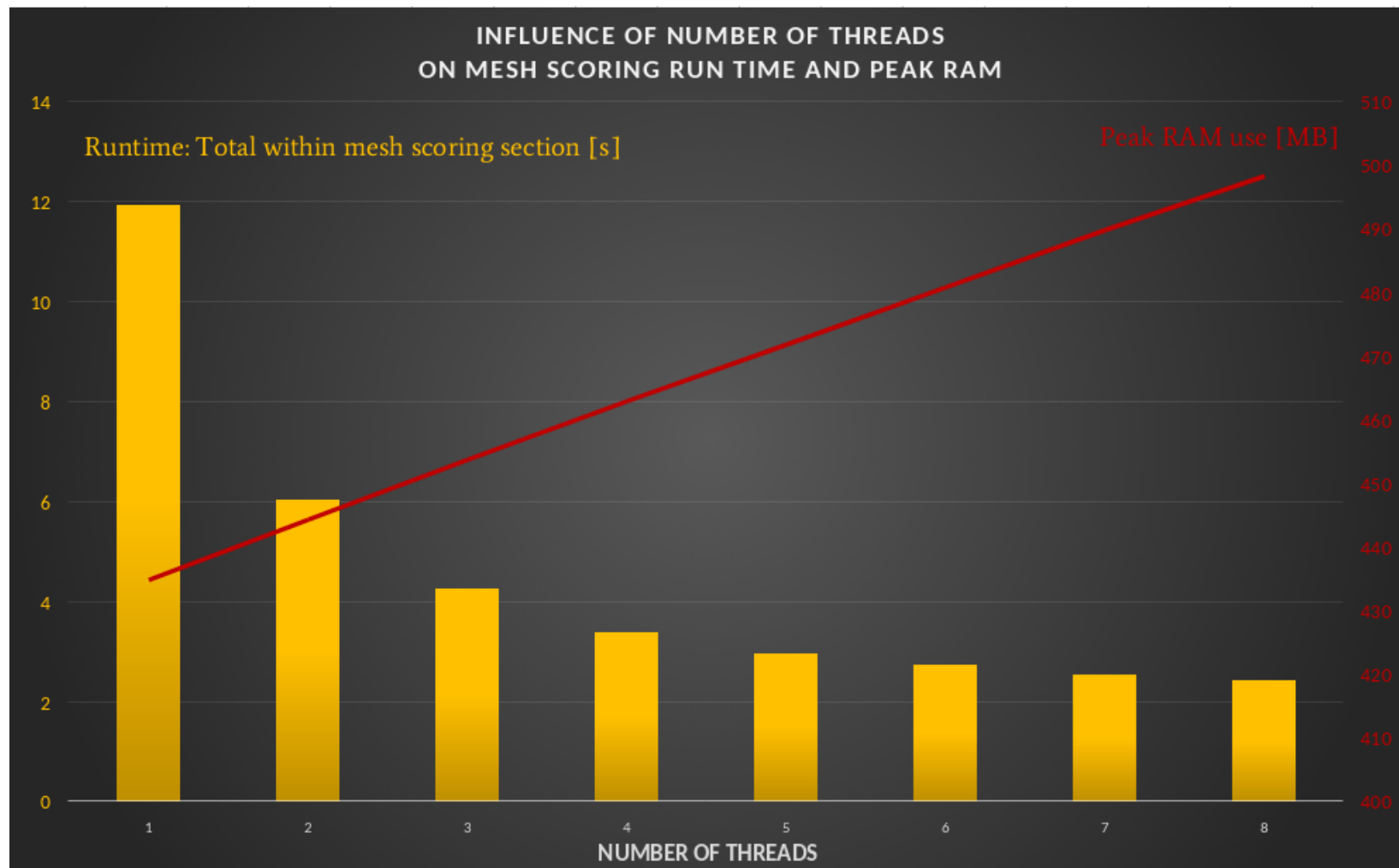
Crossing layers of water  
→ aluminum  
→ lead  
in cylinder.

FTFP\_BERT\_HP LIV.

4 000 000 primaries.

- Discrepancy with FLUKA likely due to differences in Physics models.

# MESH SCORING RUNTIME: SCALING WITH # THREADS



Scoring  
total energy  
deposition  
and fluence.

FTFP\_BERT\_HP  
LIV.

20 MeV  
neutron gun.

Cartesian  
mesh,  
**Air + Conc.**,  
60x60x45 cm,  
with  
240x240x180  
bins.

1 000 000  
events.

# CYLINDRICAL MESH SCORING RUN RESULTS: G4

Scoring total energy deposition.

FTFP\_BERT\_HP LIV.

20 GeV proton gun.

Cylindrical mesh.

Lead.

R: [0 30] cm

Phi: [0 360] deg

Z: [0 45] cm with 240x240x180 bins.

40 000 events.

Scoring fluence.

FTFP\_BERT\_HP LIV.

20 GeV proton gun.

Cylindrical mesh.

Lead.

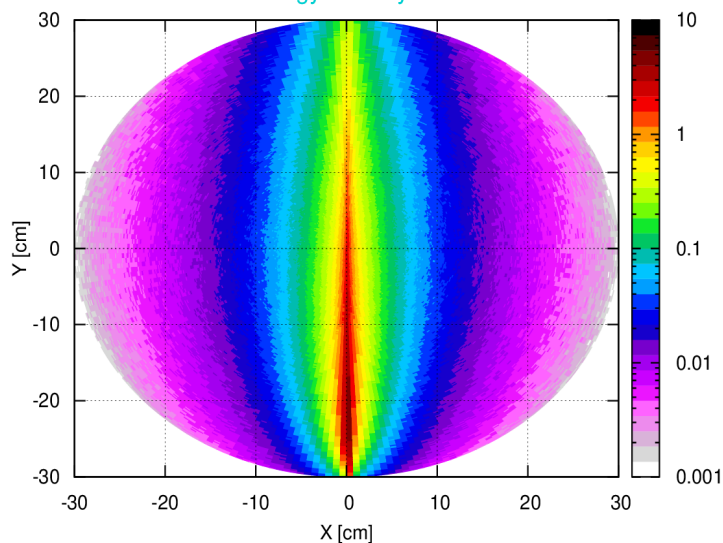
R: [0 30] cm

Phi: [0 360] deg

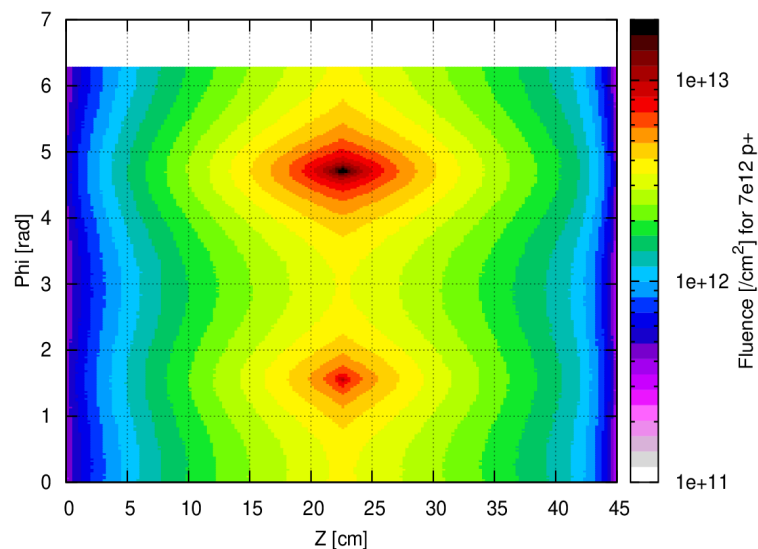
Z: [0 45] cm with 240x240x180 bins.

40 000 events.

Energy Density G4



Fluence G4



# CYLINDRICAL MESH SCORING RUN RESULTS: CUSTOM IMPLEMENTATION

Scoring total energy deposition.

FTFP\_BERT\_HP LIV.

20 GeV proton gun.

Cylindrical mesh.

Lead.

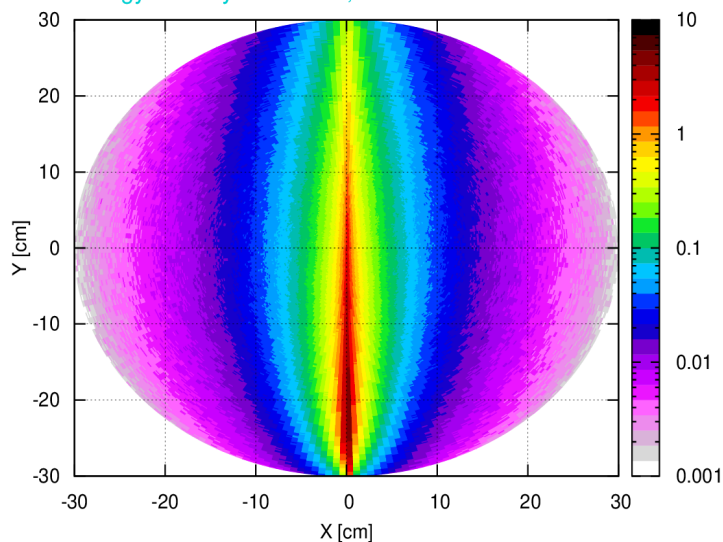
R: [0 30] cm

Phi: [0 360] deg

Z: [0 45] cm with 240x240x180 bins.

40 000 events.

Energy Density CUSTOM, NO PARALLEL WORLD



Scoring fluence.

FTFP\_BERT\_HP LIV.

20 GeV proton gun.

Cylindrical mesh.

Lead.

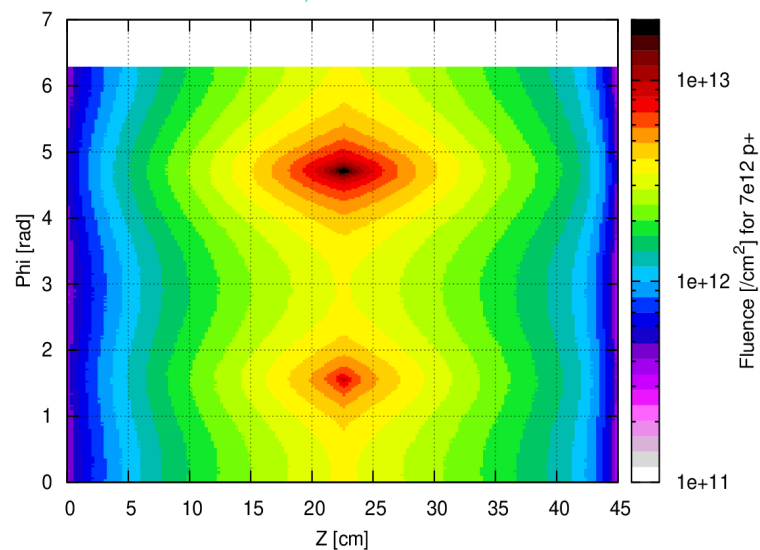
R: [0 30] cm

Phi: [0 360] deg

Z: [0 45] cm with 240x240x180 bins.

40 000 events.

Fluence CUSTOM, NO PARALLEL WORLD



# CYLINDRICAL MESH SCORING RUN RESULTS: G4 VS CUSTOM

Scoring fluence.

FTFP\_BERT\_HP LIV.

20 GeV proton gun.

Cylindrical mesh.

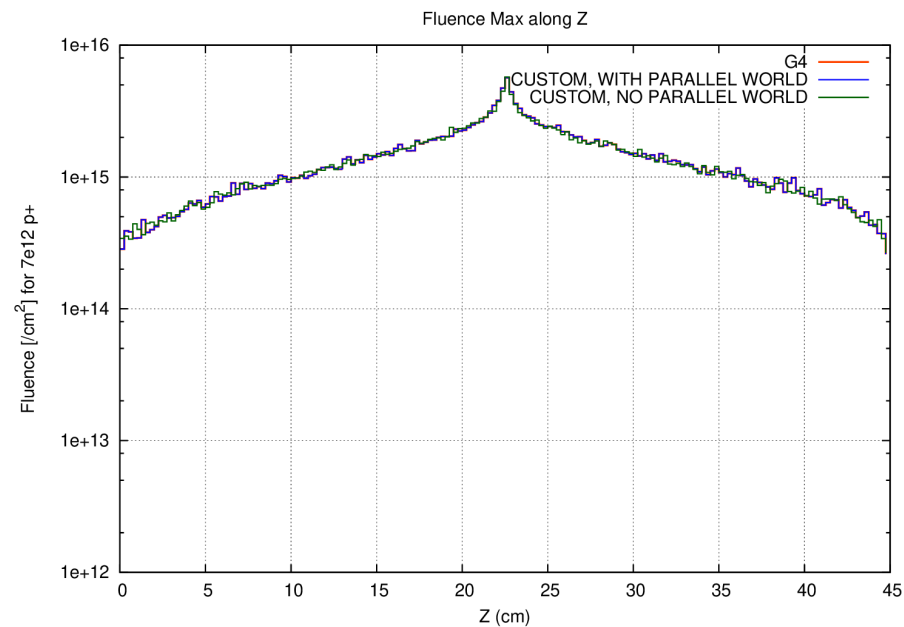
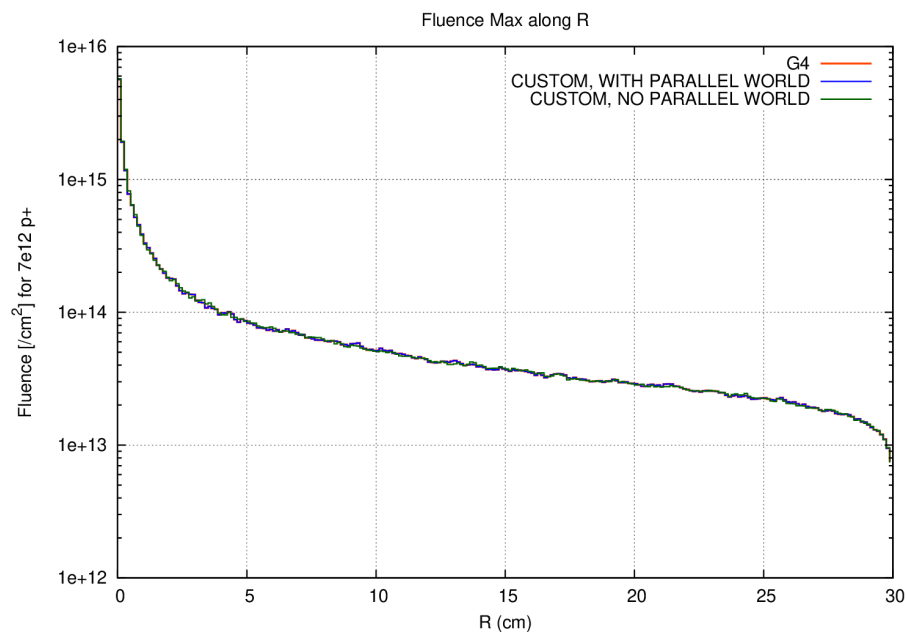
Lead.

R: [0 30] cm

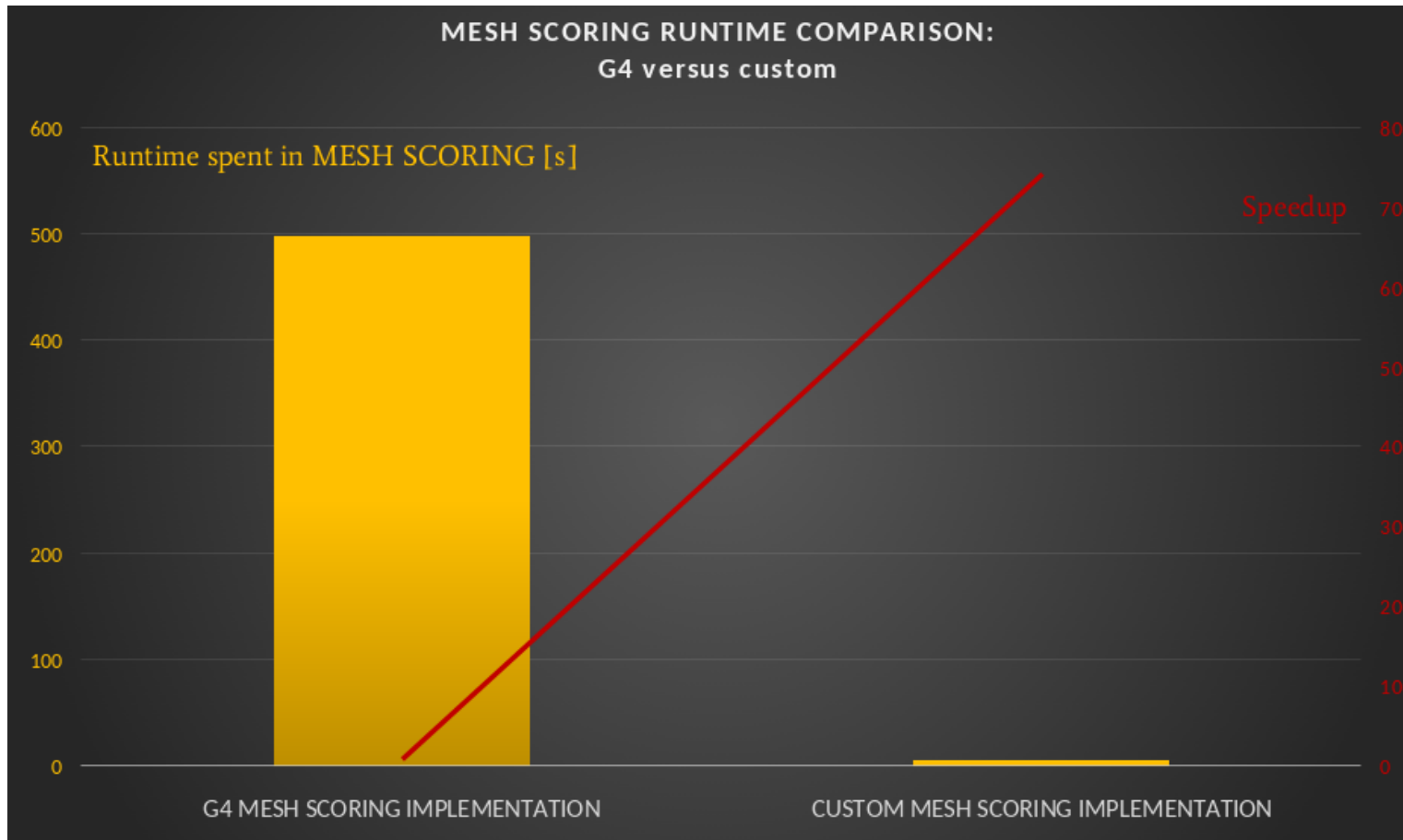
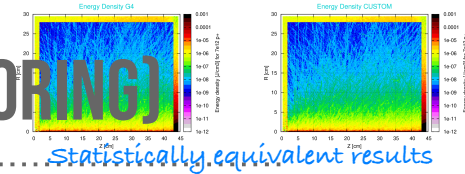
Phi: [0 360] deg

Z: [0 45] cm with 240x240x180 bins.

40 000 events.



# CYLINDRICAL MESH SCORING PERFORMANCE VS G4 (MESH SCORING)



Scoring  
total energy  
deposition  
and fluence.

FTFP\_BERT\_HP  
LIV.

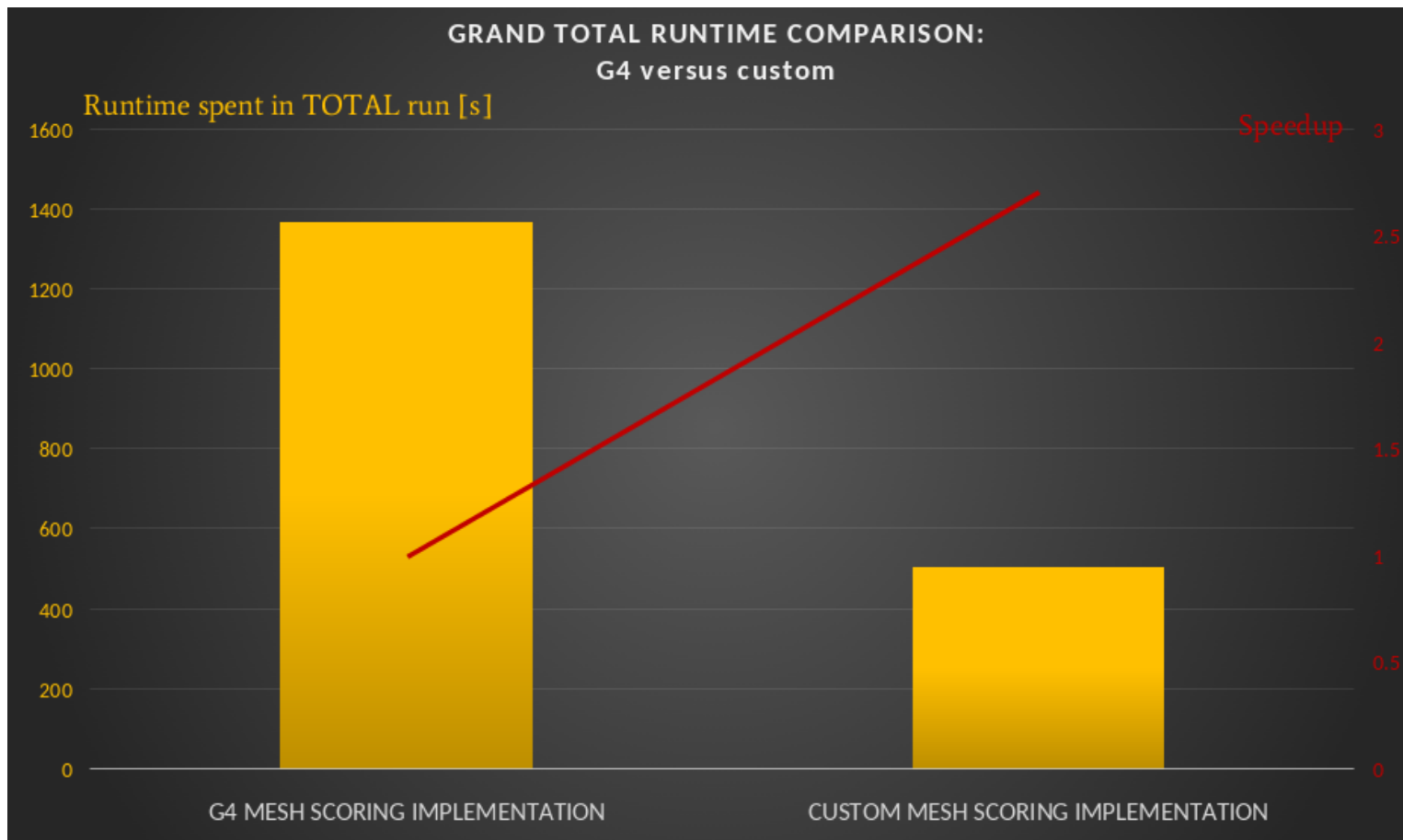
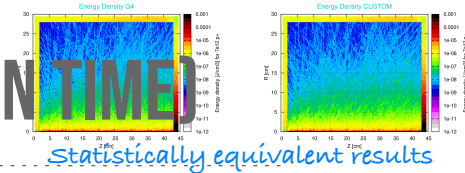
20 MeV  
neutron gun.

Cylindrical  
mesh,  
Air + Conc.,  
R: [0 30] cm  
Phi:[0 360] deg  
Z: [0 45] cm  
with  
240x240x180  
bins.

5 000 000  
events.

- ~ x74 speedup in TOTAL MESH SCORING RUN TIME (geo step splitting, saving scoring data, flushes to master...).

# CYLINDRICAL MESH SCORING PERFORMANCE VS G4 (TOTAL RUN TIME)



Scoring  
total energy  
deposition  
and fluence.

FTFP\_BERT\_HP  
LIV.

20 MeV  
neutron gun.

Cylindrical  
mesh,  
Air + Conc.,  
R: [0 30] cm  
Phi:[0 360] deg  
Z: [0 45] cm  
with  
240x240x180  
bins.

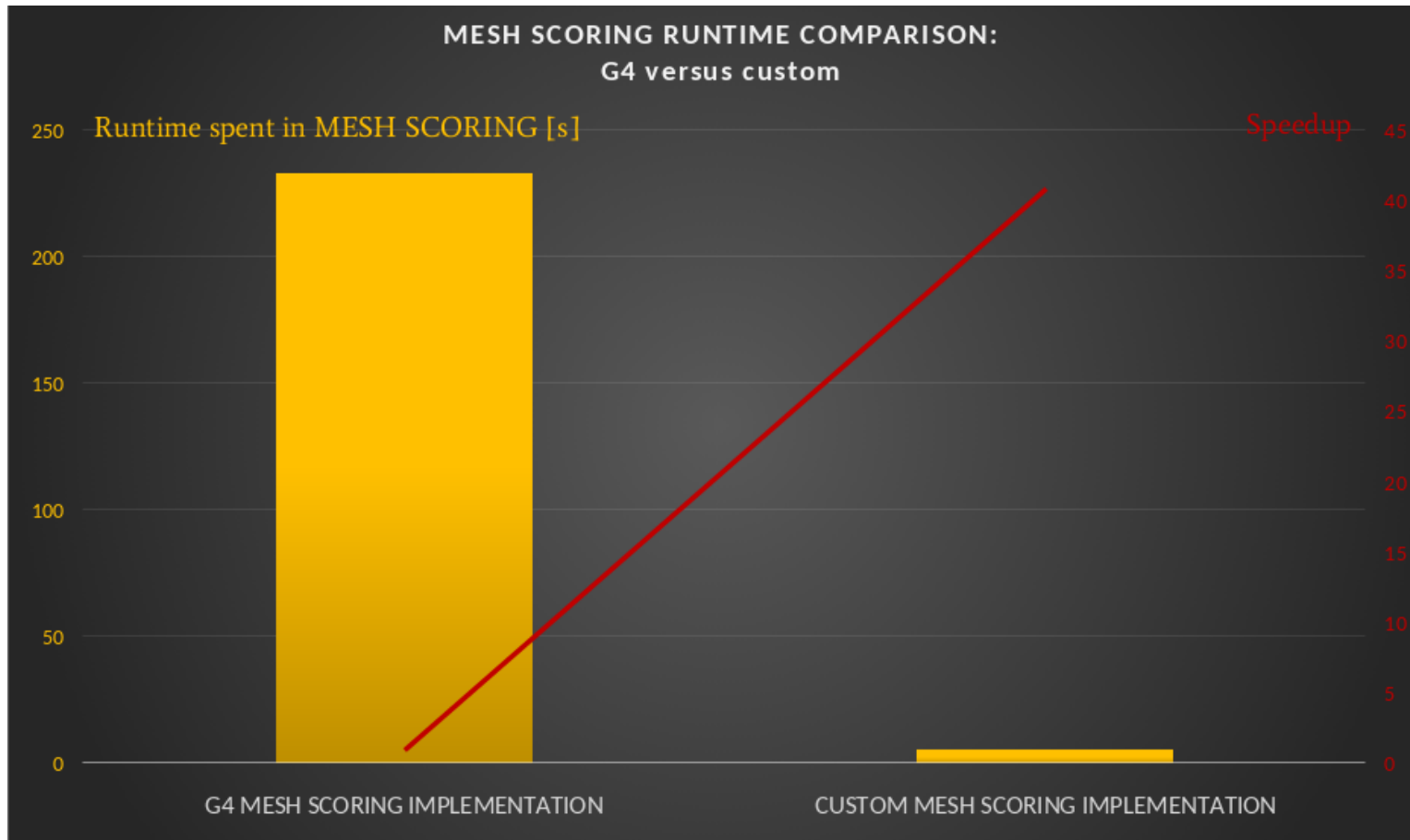
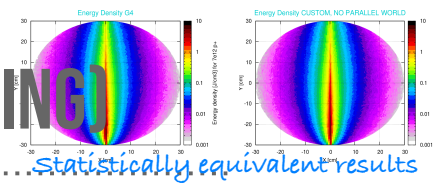
5 000 000  
events.

Peak RAM use:

- G4:  
6.6 GB
- CUSTOM  
implementation:  
535 MB

- ~ x2.7 speedup in TOTAL RUN TIME (entire moira run, including all initialization, geometry construction...).

# CYLINDRICAL MESH SCORING PERFORMANCE VS G4 (MESH SCORING)



Scoring  
total energy  
deposition  
and fluence.

FTFP\_BERT\_HP  
LIV.

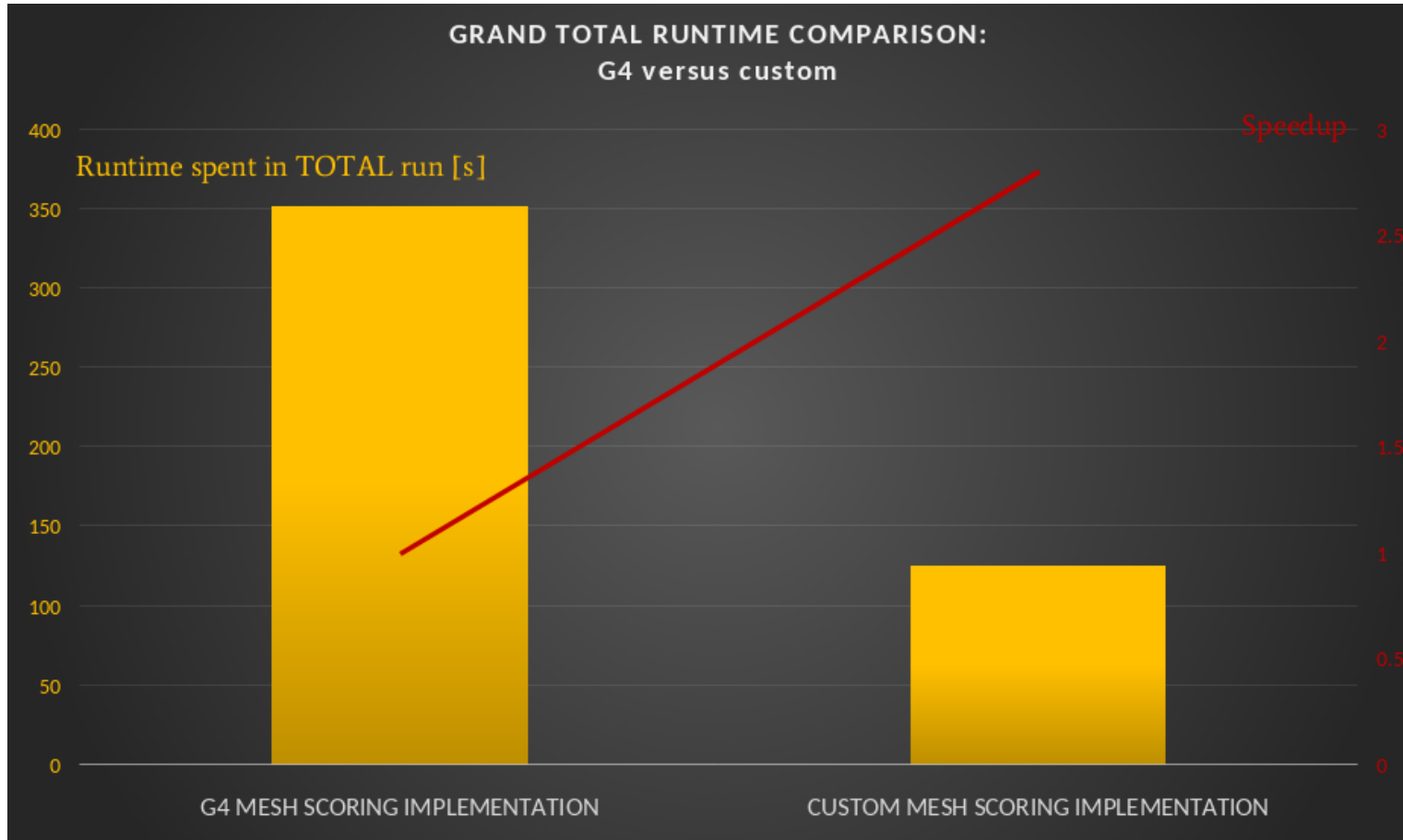
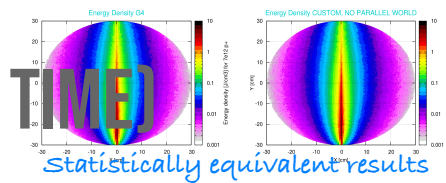
20 GeV  
proton gun.

Cylindrical  
mesh,  
Lead,  
R: [0 30] cm  
Phi:[0 360] deg  
Z: [0 45] cm  
with  
240x240x180  
bins.

400 events.

- ~ x41 speedup in TOTAL MESH SCORING RUN TIME (geo step splitting, saving scoring data, flushes to master...).

# CYLINDRICAL MESH SCORING PERFORMANCE VS G4 (TOTAL RUN TIME)



Scoring  
total energy  
deposition  
and fluence.

FTFP\_BERT\_HP  
LIV.

20 GeV  
proton gun.

Cylindrical  
mesh,  
Lead,  
R: [0 30] cm  
Phi:[0 360] deg  
Z: [0 45] cm  
with  
240x240x180  
bins.

400 events.

Peak RAM use:

- G4:  
8.7 GB
- CUSTOM  
implementation:  
539 MB

- ~ x2.8 speedup in TOTAL RUN TIME (entire moira run, including all initialization, geometry construction...).