

Benchmarking CORSIKA 8

Nikolaos Karastathis, Pranav Sampathkumar and Maximilian Reininghaus

for the CORSIKA 8 collaboration





- 10 vertical hadronic showers
- 5 45 degrees em showers
- Runtime vs energy cut for hadronic showers
- Profiling

WIT Nikos Karastathis (nikolaos.karastathis@kit.edu) – Karlsruhe Institute of Technology

Workstation specifications:

- CPU: AMD Ryzen Threadripper 2970WX 24-Core Processor Base clock speed: 3GHz Max clock speed: 4.2GHz L3 cache: 64 MB
- RAM memory: 64 GB

CORSIKA 8

Run specifications:

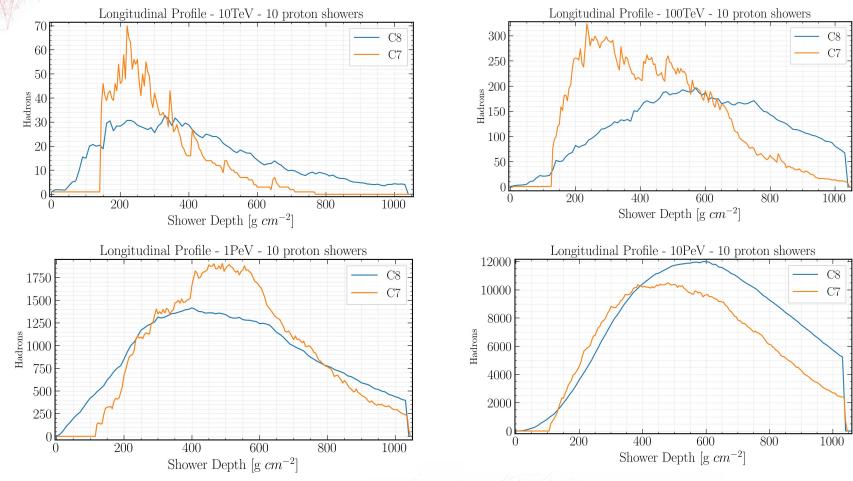
- C7: FFLAGS=-O2 CFLAGS=-O2 ./coconut – QGSJETII-04, URQMD 1.3cr
- C8: vertical_EAS example
 - logging set to WARN
 - CUTS were set to appropriate values in order to avoid crushes in PROPOSAL
 - Trackwriter is disabled

Nikos Karastathis (<u>nikolaos.karastathis@kit.edu</u>) – Karlsruhe Institute of Technology

Hadronic and EM showers were produced:

- branch: 502-examples-need-some-polishing
- commit: 94875ed461a9de73d06881ea0d83941833c99fb4
- random seeds were used
- examples: vertical_EAS.cpp, em_shower.cpp

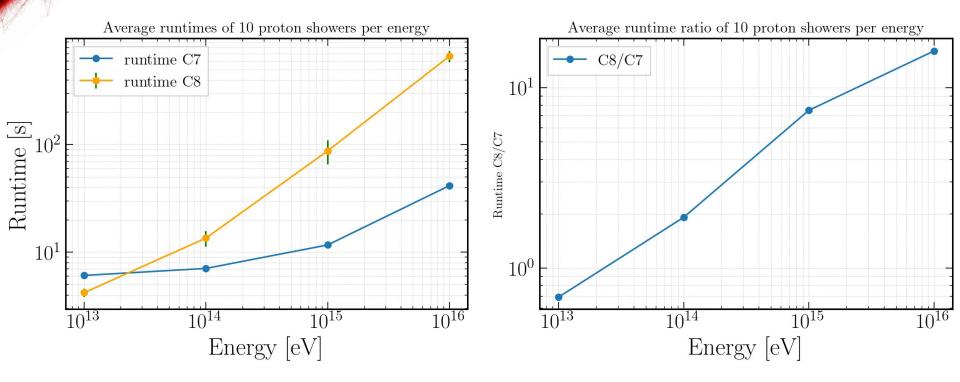
Hadronic showers



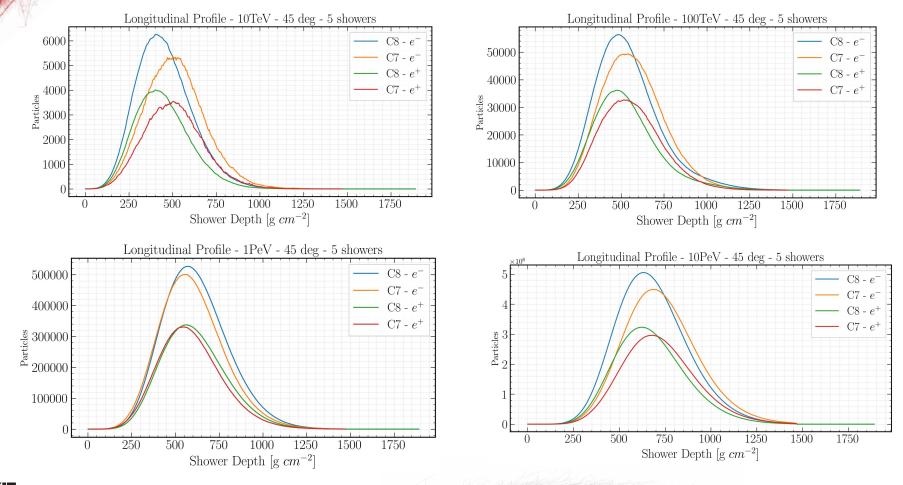
WIT Nikos Karastathis (nikolaos.karastathis@kit.edu) – Karlsruhe Institute of Technology



Hadronic showers

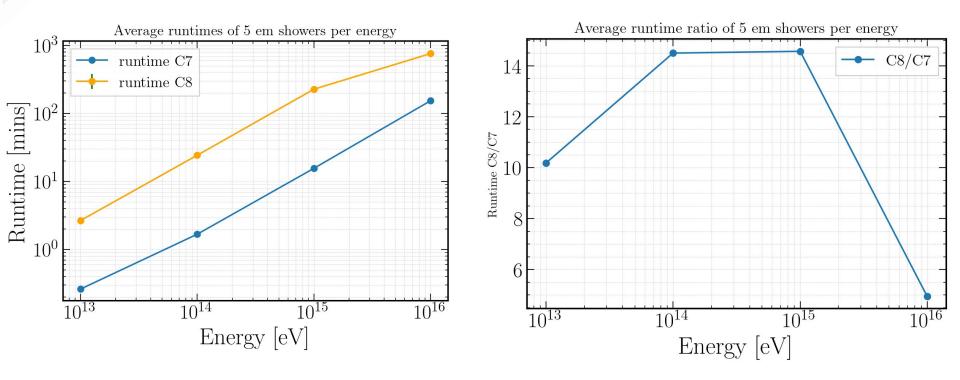


EM showers



WIT Nikos Karastathis (<u>nikolaos.karastathis@kit.edu</u>) – Karlsruhe Institute of Technology

EM showers



WIT Nikos Karastathis (<u>nikolaos.karastathis@kit.edu</u>) – Karlsruhe Institute of Technology

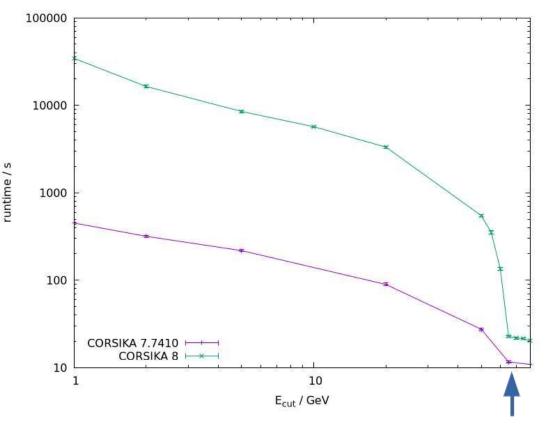
8

Runtime vs energy cut - 40 hadronic showers

primary: p 1017 eV, vertical C8: straight tracking, BetheBlochPDG C7: ELMFLG = 0, compiled with -O2

CORSIKA 8

 These showers were run by Max in his station



LE/HE transition



Profiling

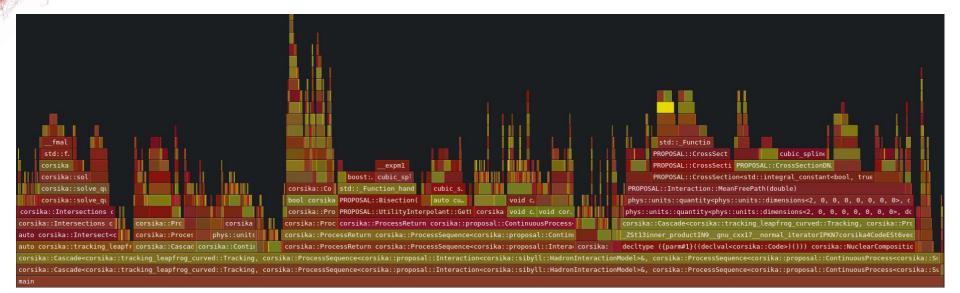
Corsika details

- branch: 502-examples-need-some-polishing
- commit: 94875e
- example: em_shower.cpp
- 100 TeV, Vertical Shower, Trackwriter disabled

Environment (Pranav's computer)

- Compilation tools: gcc=11.3.1, cmake=3.22.2
- OS: Fedora 35
- Processor: AMD Ryzen 7 PRO 3700 8-Core Processor (3.6GHz 4.4GHz)
- Performance Analysis Tools: Valgrind=3.19, perf=5.18.4, hotspot=1.3.0, kcachegrind=21.12.2

Flame Graph

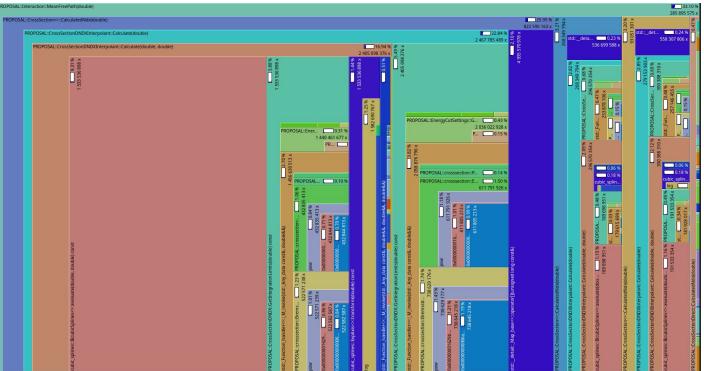


Flame Graph for CORSIKA8, this flamegraph helps us spot a few important hotspot functions, which we can analyse further, such as GetUpperLimit and MeanFreePath from PROPOSAL.

WIT Nikos Karastathis (<u>nikolaos.karastathis@kit.edu</u>) – Karlsruhe Institute of Technology

Heidelberg 2022

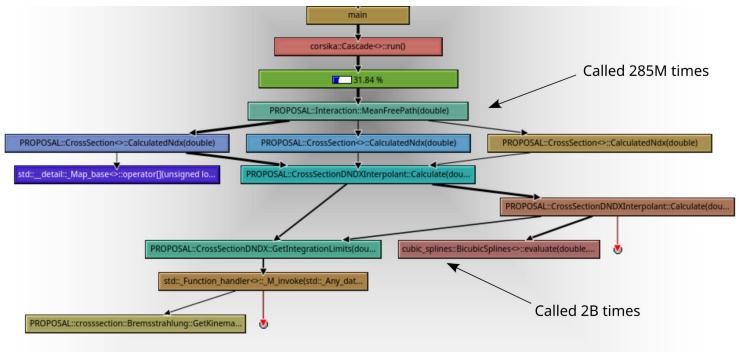
Callcache information



Callee Map for MeanFreePath, we can see that while MeanFreePath is being called 285M times, CubicSplines is being called 1.5B times, via MeanFreePath.

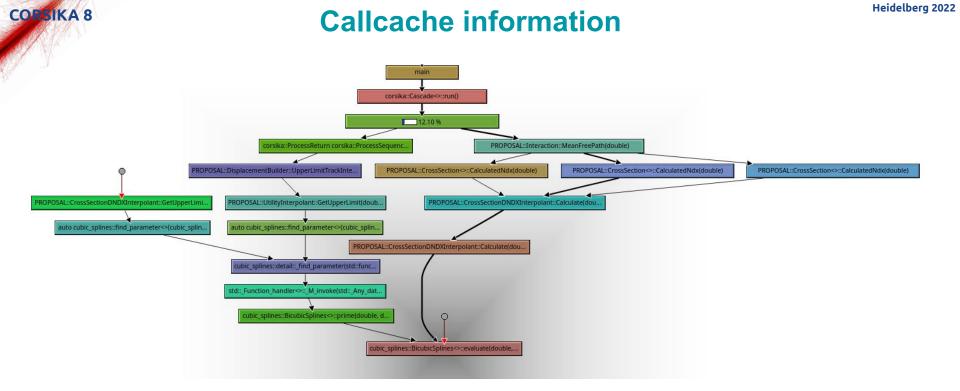
WIT Nikos Karastathis (<u>nikolaos.karastathis@kit.edu</u>) – Karlsruhe Institute of Technology

Callcache information

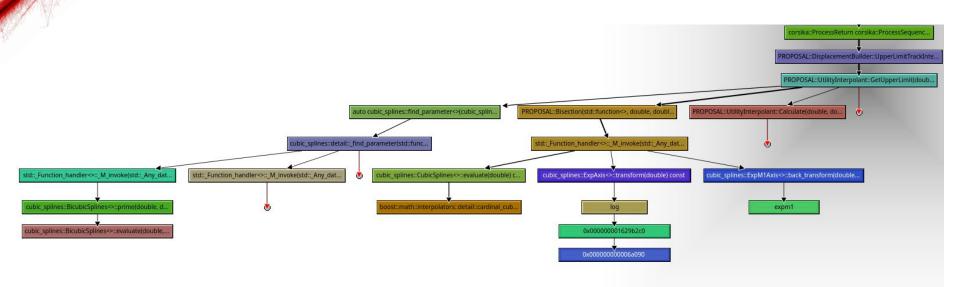


Callgraph illustrating the pathways between MeanFreePath and BiCubicSplies. The dominant pathway is highlighted.

WIT Nikos Karastathis (<u>nikolaos.karastathis@kit.edu</u>) – Karlsruhe Institute of Technology



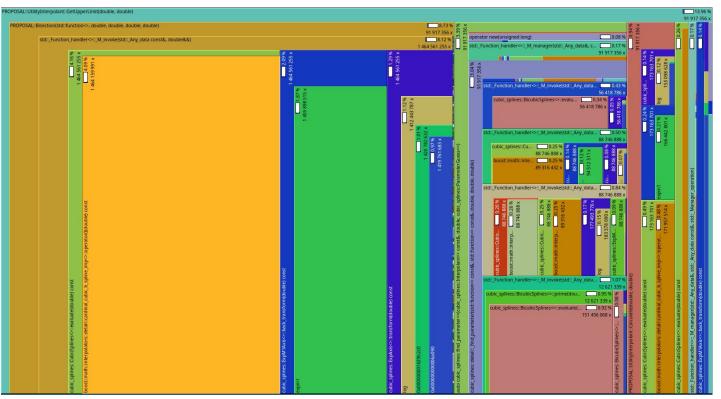
There are also other code pathways, which eventually lead to BiCubicSplies which contribute for the additional 0.5B times.



Callgraph for getUpperLimit, We dont see any immediate potential for improvement. The number of calls dont jump too much anywhere for us to spot infrastructural deficiencies.

WIT Nikos Karastathis (<u>nikolaos.karastathis@kit.edu</u>) – Karlsruhe Institute of Technology





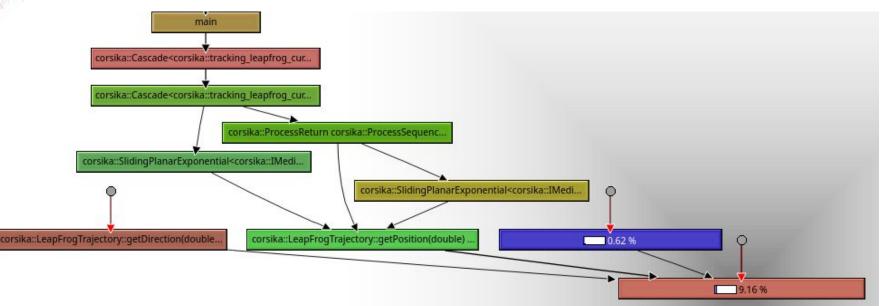
This probably calls for an algorithmic change, rather than an infrastructural change. Maybe some form of tabulation, which can reduce the calls to getUpperLimit.

WIT Nikos Karastathis (<u>nikolaos.karastathis@kit.edu</u>) – Karlsruhe Institute of Technology

| * | | 8 × | 8 × | 2 % 1 × F | * C | * × | 8 × | 5 X X |
|--------------------|---|-----------------|--|--|--|---|---|---|
| 12.0 | e operator delete(void*, unsigned long)0.0 2 void std::vector <double, std::allocator<double=""> >:: M_realloc_insert<double_const&>(gnu_cox::normal_iter0.0</double_const&></double,> | 167 99 7 % C | 0.38% x171 x171 x171 x171 x171 x171 x171 x17 | J0.3 16 44 | 31 68 | 0.2 | 0.2 | 0.21% |
| | corsika::solve_guadratic_real(long double, long double, long double, double) | 24 050 0 | 213 | 2 196 616 441 | 0.37% | 0.27% | 0.219 | 308 112 982 x |
| | 26 549 6 | 76 x | 1.00 | 21 | | | | |
| | fmal 0.18 % std://ector <d %<="" 0.07="" td=""><td></td><td></td><td></td><td></td><td></td><td></td><td></td></d> | | | | | | | |
| | | | | | | | | |
| | | | | | | | | onst |
| | | | | | | | | lateSystem const> const&) const |
| | 1, 62% 194 x 191 275 11 | | | | | | | > con |
| | 3275 | | | | | | | const |
| | | | | | | | | consider/vector/physicults:/dimensioned.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0 |
| | | | _ | | | | | ite Sys |
| | | 0.109 | | | | | | ordina |
| | | 6 225 | | | | | | anCot |
| | | 5 | | | | | 03 % | orsik |
| | | pase | | | | 05 % 82 x | 026 | ptr <c< td=""></c<> |
| | | Ited | | | | 0.05% 08112982× | 154 | ared |
| uble) | (a) | noo | | | | 308 | std:: | tdish |
| e 00 | 8 2 | IS IP | | | 96 % | Inted | 0.10 % 6 491 x | ents(s |
| qoop | 9 8 | 5 × | | | 663 3 | Sp_counted | 4 056 | uodu |
| ible, long double. | consika::andre::solve_cubic_real_analytic(long double, long double, long double, long do | 0.18 % | | | 334 | stdi: | 15 | etC or |
| 2 | g corsika=andre:solve_cubic_real_analytic(long double, long double, l | 154 051 | | ease) | SPLC | 0 % 91 × | ordin | F. |
| on g no | 0 fmal 0.47 % st <t< td=""><td></td><td></td><td>aw</td><td>stdi</td><td>onst 0.10 % 154 056 491 x</td><td>a::Co</td><td>0, 0, 1</td></t<> | | | aw | stdi | onst 0.10 % 154 056 491 x | a::Co | 0, 0, 1 |
| ble, lo | fetestexcept 0.02 % | | | 02>: | 0.11% | 154 Cons | const | 0, 0, 0, 0, 0, 1> |
| nop | feholdexcept 0.03% | | 342) | Cilod | 331.6 | const&) cons ed_ptr [| ptr 4 | 0, 0, 0, x |
| al(long do | feclearexcept 0.03% | | 2776 | X | | nt co Nared | it cor lared | 05 %) 1,05 % 982 x |
| d_rea | fesetround 0.0 feclearexcept (0.03 %) 157 084 44 fmal (0.47 %) | | 4 | | ensi. | a::Pol | tdush | ensio |
| resse | corsika::andre::solve_cubic_re (0.68 | | 5 ⁻ nub | nef. | E E | orsik ates(s | orsika ates(s | 305 |
| debi | corsika::solve_cubic_real(long (1.62 feupdateenv0.1 corsika::solve_quartic_depress (2.08 | %) | Į. | 2 | inits | ains(c | or-(co | e<(|
| Jartic | 88 041 22 corsika::Intersections corsika (5.39 | | 1 bas | d bas | phys: | etCoc | etCoc | phys: |
| | 문 · · · · · · · · · · · · · · · · · · · | (3 0 05 % | unteo | unte | ttors | here | intro | unter |
| ca::so | Enricks:andre::solve_cubic_real_analyticlong double. long do | 308 1 | td:: .Sp_counted_base* | att: Sp.conted.base4_gnu_ox:1.od.polity2 | orskat/ectorsphys.unts.idmens 005% 140: 50 c | orsika::Sphere::contains(corsika::Point const orsika::Point:getCoordinates(std::shared_pt; | orska: Pontzoperator / Cosilia: Puint constăți const orska: Pointzget Coordinates (std: shared_pri < cosilia: Coordina | ika::Vector <phys::u _Sp_counted_base*</phys::u |
| | | | | | | | | orsi) td::) |

Another, hotspot in terms of timespent, is in the LeapFrog algorithm. There is probably potential for both infrastructural and algorithmic improvements here.

WIT Nikos Karastathis (<u>nikolaos.karastathis@kit.edu</u>) – Karlsruhe Institute of Technology



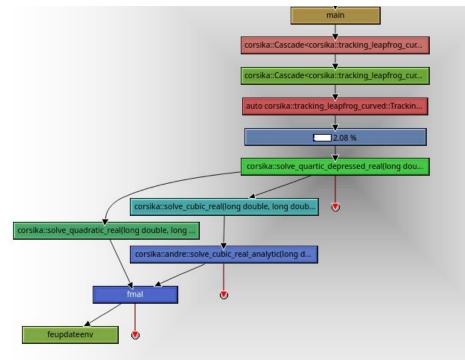
At an infrastructure level, too many calls to memory management routines like M_release. Maybe better way to allocate and deallocate memory ? This was in release mode. So these are some non-trivial allocations which couldnt be optimized by the compiler

WIT Nikos Karastathis (<u>nikolaos.karastathis@kit.edu</u>) – Karlsruhe Institute of Technology

Heidelberg 2022

Heidelberg 2022

Potential Improvement?



At an Algorithmic level, the cubic solver seems to take a decent chuck of time. We need to optimize it or avoid it altogether somehow.

WIT Nikos Karastathis (<u>nikolaos.karastathis@kit.edu</u>) – Karlsruhe Institute of Technology

Summary

- Spotted three potential places for improvement, MeanFreePath, GetUpperLimit and LeapFrog
- MeanFreePath Probably requires a infrastructural code change and a closer look into redundant calls.
- GetUpperLimit Probably requires an algorithmic change, which helps us avoid these calls
- LeapFrog Not really obvious what sort of change is required. It is probably some combination of both infrastructural and algorithmic change. So that we reduce memory allocations, and try to avoid the cubic solver or use it lesser.

Please, Playaround with these softwares yourself. A lot of insight can be gained by looking at the callgraph and the control-flow of the program.