

# Toward performance enhancements in CORSIKA 8

Profiling CORSIKA8

---

Pranav Sampathkumar

Presented at 5th CORSIKA 8 - Parallelism & Performance Meeting

January 27, 2022



### Gprof

GProf uses an hybrid instrumented and sampling approach to generate a callgraph and a flatprofile to report the time spent in each function. Gprof is unsuitable for our purposes because it doesn't account for *inclusive* costs. Gprof also doesn't profile multi threaded applications and shared libraries.

# Profilers

## Gprof

GProf uses an hybrid instrumented and sampling approach to generate a callgraph and a flatprofile to report the time spent in each function. Gprof is unsuitable for our purposes because it doesn't account for *inclusive* costs. Gprof also doesn't profile multi threaded applications and shared libraries.

## Valgrind

Valgrind runs the code in a virtual machine simulating x86 architecture. Two tools in Valgrind, *Callgrind* and *Memcheck* are very useful to us. Callgrind helps us keep track of the callgraph along with inclusive costs.

# Profilers

## Gprof

GProf uses an hybrid instrumented and sampling approach to generate a callgraph and a flatprofile to report the time spent in each function. Gprof is unsuitable for our purposes because it doesn't account for *inclusive* costs. Gprof also doesn't profile multi threaded applications and shared libraries.

## Valgrind

Valgrind runs the code in a virtual machine simulating x86 architecture. Two tools in Valgrind, *Callgrind* and *Memcheck* are very useful to us. Callgrind helps us keep track of the callgraph along with inclusive costs.

## VTune

VTune is a profiler specific to Intel. In addition to providing us with the information available in *Valgrind* and *perf*, it provides automated performance analysis which help us spot hotspots. The results provided in this presentation are gathered using VTune.

# System Information

---

## Hardware Information

- CPU Model: 11th Gen Intel(R) Core(TM) i7-1165G7 @ 2.80GHz
- Max MHz: 4700
- Cores: 8

## Software Information

- Linux Kernel v5.16
- Cmake v3.22
- gcc v11.1

## Compilation Flags

---

`-DCMAKE_BUILD_TYPE=Debug -DCMAKE_CXX_FLAGS=-pg -DCMAKE_EXE_LINKER_FLAGS=-pg -DCMAKE_SHARED_LINKER_FLAGS=-pg`

---

## Environment

---

All the profiling here are done on master(Commit ID: 01ab0f56).

Example found `./examples/corsika.cpp` is used.

Default Logging configuration (`info`) is used. All the modules are compiled using the default optimization flags. (which is `-O0` for all the modules except Pythia (`-O2`))

This uses a earth like atmosphere (Linsley US) and magnetic field (WMM Model).

Zenith and Azimuthal angle is assumed to be zero.

# Adaptive Step

## Parameters

Primary Particle:  $\mu^-$  (1e5 GeV)

## Call Stack

▼ main	100.0%	corsika.cpp
▼ corsika::ShowerAxis::ShowerAxis<corsika::IMediumPropertyModel<corsika::IMagneticField>>>	59.2%	ShowerAxis.inl
▼ _ZN5boost4math10quadrature13gauss_kronrodIdLj15ENS0_8policies6policyINS3_14	59.2%	gauss_kronrod.hpp
▶ _ZN5boost4math10quadrature13gauss_kronrodIdLj15ENS0_8policies6policyINS3_14	59.1%	gauss_kronrod.hpp

**Figure 1:** A sampling based approach to profiling: CPU samples the callstack every 10ms and then consolidates the data based on how often a function is found in the call stack

## ShowerAxis.inl

for (int i = 1; i <= steps; ++i) {	
auto const x_prev = (i - 1.) / steps;	
auto const d_prev = max_length_ * x_prev;	
auto const x = double(i) / steps;	
auto const r = boost::math::quadrature::gauss_kronrod<double, 15>::integrate(	59.2%
rho, x_prev, x, 15, 1e-9, &error);	
auto const result =	
MassDensityType(phys::units::detail::magnitude_tag, r) * max_length_;	

**Figure 2:** Using the Callstack, VTune also helps us spot lines of code which are performance hotspots



# First Profile

## Parameters

Primary Particle: Proton (1e6 GeV) Seed: 20

## Summary

Elapsed Time: 552.278s

CPU Time: 549.969s

Total Thread Count: 1

Paused Time: 0s

### Top Hotspots

This section lists the most active functions in your application. Optimizing these hotspot functions typically results in improving overall application performance.

Function	Module	CPU Time	% of CPU Time
_mcount	libc.so.6	308.343s	56.1%
std::uninitialized_copy_n<char const*, unsigned long, char*>	corsika	49.880s	9.1%
init_	corsika	9.529s	1.7%
std::shared_ptr<corsika::CoordinateSystem const>::~~shared_ptr	corsika	8.050s	1.5%
std::shared_ptr<corsika::CoordinateSystem const>::~shared_ptr	corsika	7.311s	1.3%
[Others]	N/A*	166.856s	30.3%

\*N/A is applied to non-summable metrics.

Figure 3: Summary: The Red Flags are recommendations by VTune for removing hotspots 7/20

## Call Stack

main	100.0%	corsika.cpp
▼ corsika::Cascade<corsika::tracking_leapfrog_curved::Tracking, corsika::ProcessSequence<corsika::StackInspector<corsika::Stack<corsika::CombinedStackImpl<corsika::CombinedStackImpl<	93.8%	Cascade.inl
▶ corsika::Cascade<corsika::tracking_leapfrog_curved::Tracking, corsika::ProcessSequence<corsika::StackInspector<corsika::Stack<corsika::CombinedStackImpl<corsika::CombinedStackImpl<	54.9%	Cascade.inl
▶ corsika::Stack<corsika::CombinedStackImpl<corsika::CombinedStackImpl<corsika::CombinedStackImpl<corsika::VectorStackImpl, corsika::node::GeometryData<corsika::Environment<cor	38.7%	Stack.inl
▶ corsika::Stack<corsika::CombinedStackImpl<corsika::CombinedStackImpl<corsika::CombinedStackImpl<corsika::VectorStackImpl, corsika::node::GeometryData<corsika::Environment<cor	0.1%	Stack.inl

**Figure 4:** Shows the time spend in various sections of the code. We see the time split between logging the stack and performing the cascade



## Cascade.inl Line 69, 79

while (!stack_.isEmpty()) {	0.0%
CORSIKA_LOG_TRACE("Stack: {}", stack_.asString());	38.7%
count_++;	
auto pNext = stack_.getNextParticle();	0.1%
CORSIKA_LOG_TRACE(	0.0%
"==== next particle : count={}, pid={}"	
", stack entries={}"	
", stack deleted={}",	
count_, pNext.getPID(), stack_.getEntries(), stack_.getErased());	
step(pNext);	54.9%
sequence_.doStack(stack_);	0.0%
}	

## Cascade.inl Line 350

inline ProcessReturn Cascade<TTracking, TProcessList, TOutput, TStack>::interaction( stack_view_type& view, FourMomentum const& projectileP4, NuclearComposition const& composition, CrossSectionType const initial_cross_section) {	
CORSIKA_LOG_DEBUG("collide");	
// one option is that cross section is now smaller (less // probability for collision) than it was before the step, thus, // no interaction might actually occur and is allowed	
UniformRealDistribution<CrossSectionType> uniDist(initial_cross_section);	0.0%
CrossSectionType const sample_process_by_cx = uniDist(rng_);	0.0%
auto const returnCode = sequence.selectInteraction(view, projectileP4, composition, rng_, sample_process_by_cx);	11.3%
if (returnCode != ProcessReturn::Interacted) { CORSIKA_LOG_DEBUG("Particle did not interact!"); }	
setEventType(view, history::EventType::Interaction); return returnCode; }	0.1%

## ProcessSequence.inl Line 131

if constexpr (is_process_v<process2_type>) { // to protect from further compiler	
// errors if process2_type is invalid	
if constexpr (process2_type::is_process_sequence) {	
ret  = B_.doContinuous(particle, vT, limitId);	54.4% 
} else if constexpr (is_continuous_process_v<process2_type>) {	
// interface checking on TProcess2	
static_assert(	
has_method_doContinuous_v<TProcess2, ProcessReturn, TParticle&, TTrack&>	
has_method_doContinuous_v<TProcess2, ProcessReturn, TParticle&,	
TTrack const&>	
has_method_doContinuous_v<TProcess2, ProcessReturn, TParticle const&,	
TTrack const&>,	
"TDerived has no method with correct signature \"ProcessReturn \"	
\"doContinuous(TParticle [const]&,TTrack [const]&,bool)\" required for \"	
\"ContinuousProcess<TDerived>. \");	
ret  = B_.doContinuous(particle, vT,	3.7% 
limitId == ContinuousProcessIndex(IndexProcess2));	
}	
}	

## CombinedStack.inl Line 66

inline std::string CombinedParticleInterface<TParticleInterfaceA, TParticleInterfaceB,	0.2%
TStackIterator>::asString() const {	
return fmt::format("[[{}]] [{}]]", pi_a_type::asString(), pi_b_type::asString());	65.9%
}	0.0%

**Figure 5:** Memory management for strings is slowing down the code significantly

# Memory Profile

## Summary

### Elapsed Time : 2163.763s

Allocation Size: 27.5 GB  
Deallocation Size: 27.5 GB  
Allocations: 178,132,088  
Total Thread Count: 1  
Paused Time : 0s

### Top Memory-Consuming Functions

This section lists the most memory-consuming functions in your application.

Function	Memory Consumption	Allocation/Deallocation Delta	Allocations	Module
fmt::v7::detail::vformat[abi:cxx11]	15.5 GB	0.0 B	172,684,691	corsika
corsika::Stack<corsika::CombinedStackImpl<corsika::CombinedStackImpl<corsika::CombinedStackImpl<corsika::VectorStackImpl, corsika::node::GeometryData<corsika::Environment<corsika::IMediumPropertyModel<corsika::IMagneticFieldModel<corsika::IMediumModel>>>>, corsika::weights::WeightData>, corsika::history::HistoryData<corsika::history::Event>>, corsika::setup::detail::StackWithHistoryInterface, corsika::history::HistorySecondaryProducer>>>::asString[abi:cxx11]	11.4 GB	0.0 B	1,552,812	corsika ↗
PROPOSAL::Interaction::Rates	171.8 MB	0.0 B	875,724	corsika
corsika::operator<<<phys::units::dimensions<(int)1, (int)0, (int)0, (int)0, (int)0, (int)0, (int)0, (int)0, (int)0>>>	94.6 MB	0.0 B	184,484	corsika ↗
__gnu_cxx::new_allocator<std::pair<unsigned long, double>>>::allocate	55.8 MB	0.0 B	1,163,332	corsika ↗
[Others]	290.1 MB	29.5 MB	1,671,045	N/A*

\*N/A is applied to non-summable metrics.

**Figure 6:** Memory Profiles are usually done in an instrumental way, where the instruction pointer for each allocation event is stored along with the call sequence

## Stack

Module / Function / Function Stack	Allocation/Deallocation Delta	Allocation Size	Deallocation Size	Allocations ▼
▼ corsika	29.4 MB	27.5 GB	27.5 GB	178,132,058
▶ fmt::v7::detail::vformat[abi:cxx11]	0 B	15.5 GB	15.5 GB	172,684,691
▶ corsika::Stack<corsika::CombinedStackImpl<corsik	0 B	11.4 GB	11.4 GB	1,552,812
▶ __gnu_cxx::new_allocator<std::pair<unsigned long	0 B	55.8 MB	55.8 MB	1,163,332
▶ PROPOSAL::Interaction::Rates	0 B	171.8 MB	171.8 MB	875,724
▶ __gnu_cxx::new_allocator<corsika::history::Second	0 B	34.2 MB	34.2 MB	369,304

**Figure 7:** Significant number of allocations for string manipulations



## Core.h Line 2076

```
FMT_INLINE std::basic_string<Char> format(const S& format_str, Args&&... args) {  
    const auto& vargs = fmt::make_args_checked<Args...>(format_str, args...);  
    > return detail::vformat(to_string_view(format_str), vargs);  
}
```

## Stack.inl Line 327

for (unsigned int iPart = 0; iPart != getSize(); ++iPart) {	
const_stack_iterator_type itPart("this, iPart);	
str += fmt::format("{}{}{}", new_line, itPart.asString(),	11.4 GB
(deleted_[itPart.getIndex()] ? " [deleted]" : ""));	
new_line = "\n    ";	
}	

# Second Profile (O3 Optimized)

## Parameters

Primary Particle: Proton (1e6 GeV) Seed: 20

## Summary

Elapsed Time: 880.234s

CPU Time: 876.980s  
Total Thread Count: 1  
Paused Time: 0s

### Top Hotspots

This section lists the most active functions in your application. Optimizing these hotspot functions typically results in improving overall application performance.

Function	Module	CPU Time	% of CPU Time
pow	libm.so.6	436.502s	49.8%
func@0x79684	libm.so.6	129.550s	14.8%
PROPOSAL::crosssection::EpairKelnerKokoulinPetrushin::FunctionToIntegral	corsika	41.579s	4.7%
PROPOSAL::crosssection::EpairForElectronPositron::FunctionToIntegral	corsika	34.610s	3.9%
expf64	libm.so.6	29.601s	3.4%
[Others]	N/A*	205.138s	23.4%

\*N/A is applied to non-summable metrics.

Figure 8: Summary: Surprisingly the O3 optimization flag is slower than the default optimization

# Call Stack

main	100.0%	corsika.cpp
run	99.2%	Cascade.inl
step	95.9%	Cascade.inl
getWeightedSum<corsika::Cascade<corsika::tracking_leapfrog_curved::Tracking, corsika::ProcessSequence<corsika::StackInspector<corsika::Stack<corsika::CombinedStackImpl<corsika::C...	93.3%	NuclearComposi...
inner_products<_gnu_cxx::__normal_iterator<const corsika::Code*, std::vector<corsika::Code>, >, _gnu_cxx::__normal_iterator<double const*, std::vector<double, std::allocator<double>>	93.3%	std_numeric.h
operator()<corsika::Code, double>	93.3%	NuclearComposi...
operator()	93.3%	Cascade.inl
getCrossSection<corsika::StackIteratorInterface<corsika::CombinedStackImpl<corsika::CombinedStackImpl<corsika::CombinedStackImpl<corsika::VectorStackImpl, corsika::node::G...	93.3%	ProcessSequen...
getCrossSection<corsika::StackIteratorInterface<corsika::CombinedStackImpl<corsika::CombinedStackImpl<corsika::CombinedStackImpl<corsika::VectorStackImpl, corsika::node::G...	93.3%	ProcessSequen...
corsika::ProcessSequence<corsika::ProcessSequence<corsika::pythia8::Decay&, corsika::sbyll::Decay&, (int)0, (int)2, (int)1>&, corsika::ProcessSequence<corsika::ParticleCut<	93.3%	ProcessSequen...
corsika::ProcessSequence<corsika::ParticleCut<corsika::SubWriter<corsika::EnergyLossWriter<corsika::EnergyLossWriterParquet<(unsigned long)1>>>&, corsika::ProcessS...	93.3%	ProcessSequen...
corsika::ProcessSequence<corsika::proposal::Interaction&, corsika::ProcessSequence<corsika::BetheBlochPDG<corsika::SubWriter<corsika::EnergyLossWriter<corsika::En...	93.3%	ProcessSequen...
corsika::proposal::Interaction::getCrossSection<corsika::StackIteratorInterface<corsika::CombinedStackImpl<corsika::CombinedStackImpl<corsika::CombinedStackImpl<c...	93.3%	Interaction.inl
corsika::proposal::ProposalProcessBase::getCalculator<corsika::StackIteratorInterface<corsika::CombinedStackImpl<corsika::CombinedStackImpl<corsika::CombinedSt...	93.2%	ProposalProces...
corsika::proposal::Interaction::buildCalculator	93.2%	Interaction.inl
std::function<std::vector<std::shared_ptr<PROPOSAL::CrossSectionBase>, std::allocator<std::shared_ptr<PROPOSAL::CrossSectionBase>>>> (PROPOSAL::Mediu...	92.8%	std_function.h
M_invoke	24.2%	std_function.h
_invoke_r<std::vector<std::shared_ptr<PROPOSAL::CrossSectionBase>, >, lambda<PROPOSAL::Medium&, corsika::units::si::HEPEnergyType>&>&, PROPOS...	24.2%	invoke.h
_invoke_impl<std::vector<std::shared_ptr<PROPOSAL::CrossSectionBase>, >, lambda<PROPOSAL::Medium&, corsika::units::si::HEPEnergyType>&>&, PR...	24.2%	invoke.h
operator()	24.2%	ProposalProces...
PROPOSAL::GetStdCrossSections<PROPOSAL::EMinusDef, PROPOSAL::Medium, std::shared_ptr<PROPOSAL::EnergyCutSettings const>, bool>	24.2%	ParticleDefaultC...
PROPOSAL::DefaultCrossSections<PROPOSAL::EMinusDef>::Get<PROPOSAL::Medium, std::shared_ptr<PROPOSAL::EnergyCutSettings const>, t...	24.2%	ParticleDefaultC...
PROPOSAL::DefaultCrossSections<PROPOSAL::EMinusDef>::Append<std::vector<std::shared_ptr<PROPOSAL::CrossSectionBase>, std::allocat...	24.2%	ParticleDefaultC...
PROPOSAL::append_cross<std::vector<std::shared_ptr<PROPOSAL::CrossSectionBase>, std::allocator<std::shared_ptr<PROPOSAL::CrossSec...	24.2%	ParticleDefaultC...
PROPOSAL::append_cross<std::vector<std::shared_ptr<PROPOSAL::CrossSectionBase>, std::allocator<std::shared_ptr<PROPOSAL::CrossS...	24.0%	ParticleDefaultC...
PROPOSAL::append_cross<std::vector<std::shared_ptr<PROPOSAL::CrossSectionBase>, std::allocator<std::shared_ptr<PROPOSAL::Cross...	15.7%	ParticleDefaultC...
PROPOSAL::append_cross<std::vector<std::shared_ptr<PROPOSAL::CrossSectionBase>, std::allocator<std::shared_ptr<PROPOSAL::Cr...	15.7%	ParticleDefaultC...
PROPOSAL::make_crosssection<PROPOSAL::crosssection::PhotoAbramowiczLevinLevyMaor97&, PROPOSAL::EMinusDef&, PROPO...	15.7%	CrossSectionBul...
std::make_unique<PROPOSAL::CrossSectionInterpolator<PROPOSAL::crosssection::PhotoAbramowiczLevinLevyMaor97&, PROPOS...	15.7%	unique_ptr.h
PROPOSAL::CrossSectionInterpolator<PROPOSAL::crosssection::PhotoAbramowiczLevinLevyMaor97&, PROPOSAL::crosssection...	15.7%	CrossSectionInt...
PROPOSAL::CrossSection<std::integral_constant<bool, (bool)1>, std::integral_constant<bool, (bool)0>>>::CrossSection<PROPOS...	15.7%	CrossSection.h
PROPOSAL::detail::build_dndx<PROPOSAL::crosssection::PhotoAbramowiczLevinLevyMaor97>	15.7%	CrossSection.h
PROPOSAL::make_dndx<PROPOSAL::crosssection::PhotoAbramowiczLevinLevyMaor97&, PROPOSAL::ParticleDef&, PRO...	15.7%	CrossSectionDN...
std::make_unique<PROPOSAL::CrossSectionDNDXInterpolator, PROPOSAL::crosssection::PhotoAbramowiczLevinLevyMa...	15.7%	unique_ptr.h
PROPOSAL::CrossSectionDNDXInterpolator<PROPOSAL::crosssection::PhotoAbramowiczLevinLevyMaor97&, PROPOSAL::ParticleDef&, PRO...	15.7%	CrossSectionDN...
cubic_splines::interpolant-cubic_splines::BicubicSplines<double>, cubic_splines::BicubicSplines<double>::Definition>	15.7%	interpolant.h
cubic_splines::BicubicSplines<double>::BicubicSplines	15.7%	

**Figure 9:** Call Stack: With the optimizations for memory management out of the way, we see time being spent in PROPOSAL

## Interaction.inl Line 38

```
auto c = p_cross->second(media.at(comp.getHash()), emCut);
```

92.8%

## ProposalProcessBase.hpp Line 49

```
static auto cross_builder =  
    [](PROPOSAL::Medium& m,  
        corsika::units::si::HEPEnergyType  
        emCut) { //!< Stochastic losses smaller than the given cut  
                //!< will be handled continuously.  
        auto p_cut = std::make_shared<const PROPOSAL::EnergyCutSettings>(  
            0.5 * emCut / 1_MeV, 1, false);  
        return PROPOSAL::GetStdCrossSections(T(), m, p_cut, true);  
    };
```

24.2%

0s

# To Do

- Computation size tradeoff in adaptive stepping (maybe reimplement Gauss-Kronrod ? (Boost vs GSL))
- Alternate way to handle strings?
- Some immediate places to parallelize! (Like Log Stack)
- PROPOSAL: pow being used right ? (Data types/ Check for overusage)