Preliminary work on corsika optimization

L. Arrabito[1], J. Bregeon[1],
P. Langlois[2], D. Parello[2], G. Revy[2]
[1]LUPM CNRS-IN2P3 France
[2]DALI UPVD-LIRMM France

F2F REPRISES, 21st Juin 2018, CC-IN2P3 Villeurbanne

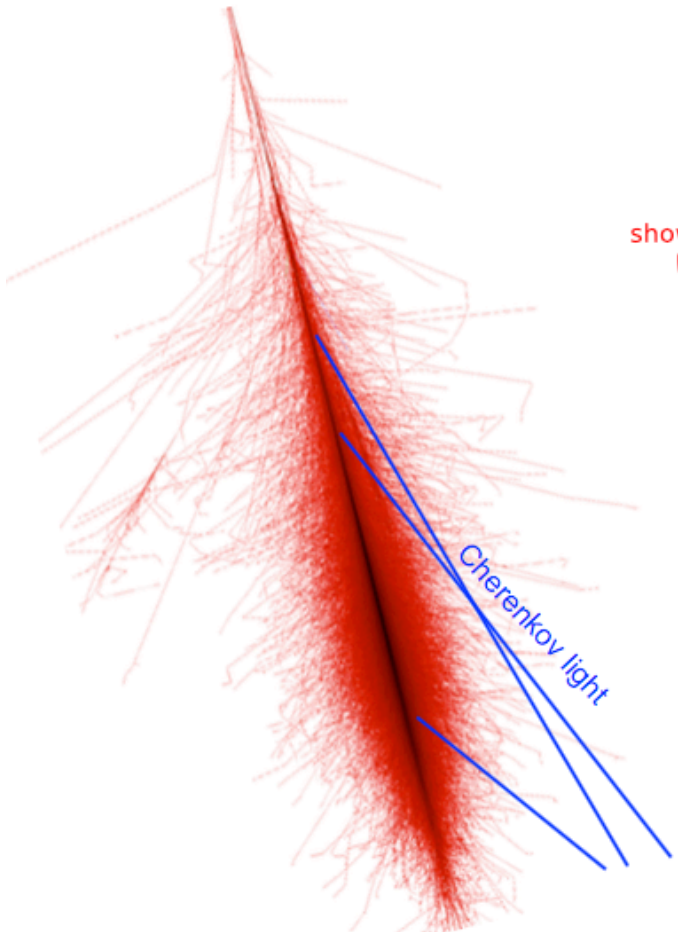
Plan

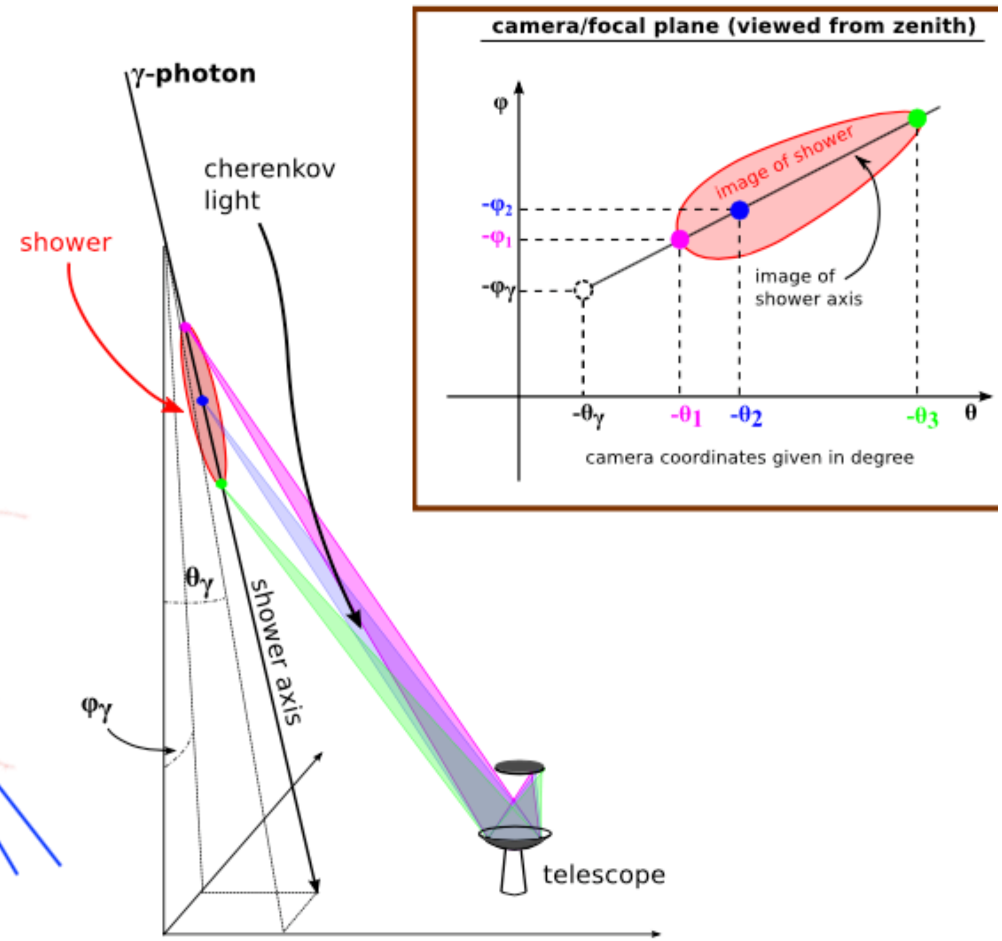
- Brief introduction to corsika
- Corsika profiling
- Compiler optimization tests
- First manual optimizations
- Next steps and conclusions

Introduction to corsika

- Detailed simulation of showers initiated by high energy cosmic rays
- Initially developed for the Kaskade experiment (since 1990 at KIT)
- Today is widely used by several ‘cosmic rays’ communities
- 900 users from 57 countries
- > 1900 citations
- https://www.ikp.kit.edu/corsika/

corsika for CTA





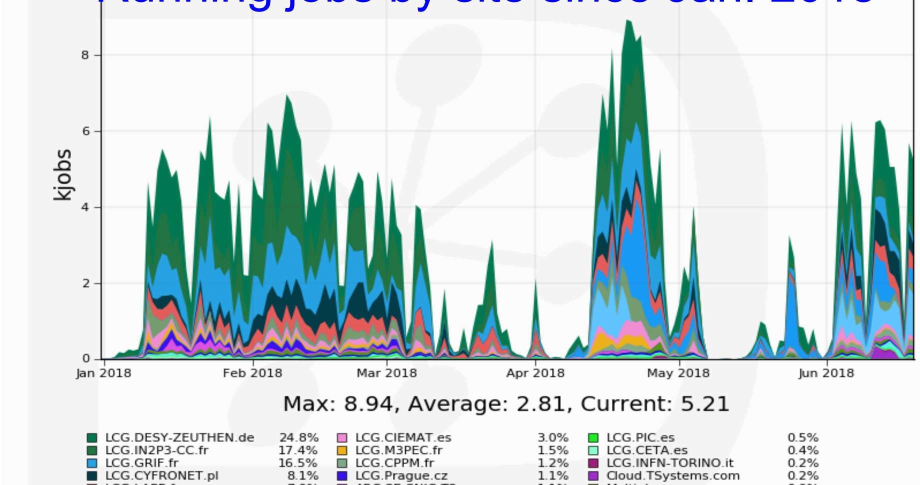
The software

- Main program (Fortran)
  - A single source of about 70k lines of code
- Customized external packages for electromagnetic and hadron interactions (Fortran)
  - EGS4, FLUKA, UrQMD, GHEISHA, QGSJET, EPOS-LHC, DPMJET, SIBYLL
- IACT/atmo package (written in C)
  - Extension to corsika to implement arrays of Cherenkov telescopes
  - Use of external atmospheric models
  - Propagation of light in the atmosphere with refraction
- Total of > 10^5 lines of code
- Many person-years of development
- Project of full re-writing in modular C++ just started
  - Open source project

Motivations to improve corsika performances

- MC simulations in CTA are the most CPU consuming task
  - 70% of CPU spent in corsika (shower development)
  - 30% of CPU spent in telescope simulation
- Massive MC simulations run on the grid since 7 years to assess CTA design
- During CTA operations MC simulations will be periodically run to calculate the Instrument Response Functions

Running jobs by site since Jan. 2018



← 8000 jobs

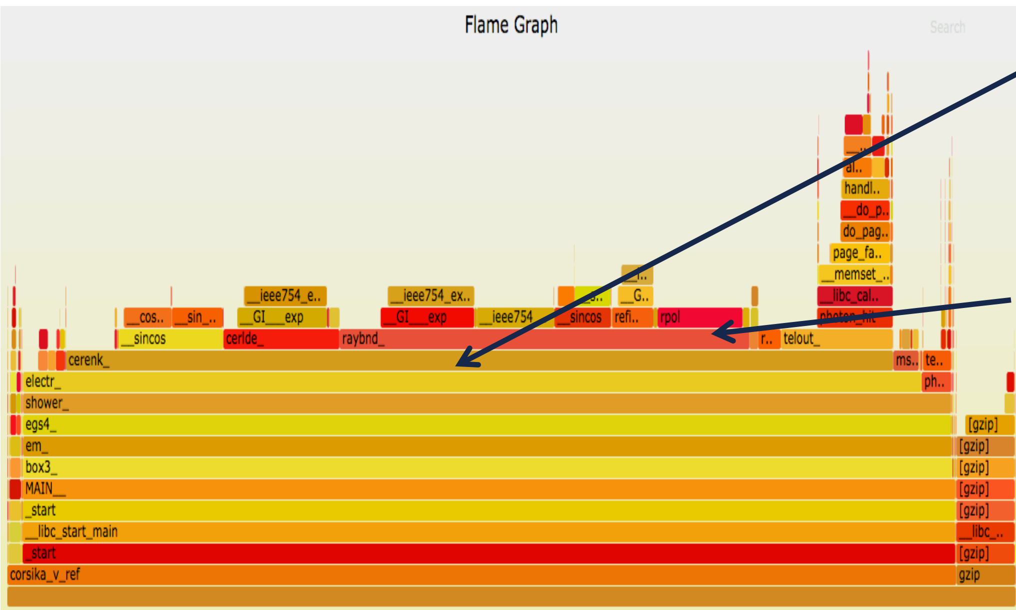
- 6000-8000 concurrent jobs
- > 125 M HS06 CPU hours since Jan. 2018

Corsika profiling with Linux perf

- Profiler tool for Linux based systems
- Used the sampling method (perf record/report), based on the 'cycles' event and using the call graph option
- Running on a dedicated server
  - x86_64
  - Intel(R) Xeon(R) CPU E5-2650 v4 @ 2.20GHz
  - CentOS Linux release 7.4.1708 (Core)
  - Compiled with: -O2 –funroll-loops
- Use 'standard' corsika input parameters (the same as in production)

Profiling results

Linux perf + FlameGraph

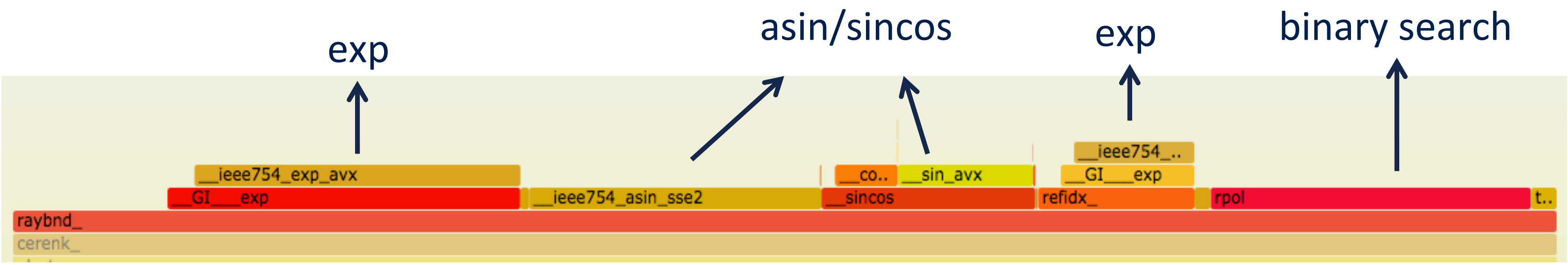


- 90% of CPU in *CERENK* subroutine and below
  - Cherenkov photon production
  - Part of corsika 'core'
- 50% of CPU in *raybnd* function and below
  - Propagation of cherenkov photon in the atmosphere with refraction correction
  - Part of IACT/atmo package

- Compatible results obtained with different profiling tools
  - https://poormansprofiler.org/ (based on gdb)
  - valgrind

Profiling results

- Zoom on raybnd (50% CPU)



- Most of the CPU spent in mathematical functions and atmospheric/refraction profile interpolation
  - 35% exp (used for atmospheric profile interpolation)
  - 35% sincos/asin
  - 20% binary search for refraction tables interpolation
- Very frequently called, once per photon bunch
  - About 160k photon bunches per shower (in our tests)
- Photon bunches are treated independently
  - Possible vectorization?
- Choose to start optimizing the raybnd function

Optimization strategy

- Test automatic optimizations by compiler
  - We did not expect significant gains
- Apply manual transformations
  - At algorithmic level
    - *e.g.* Testing different atmospheric interpolation schemes
  - Code refactoring
  - Exploiting the micro-architecture capabilities
    - Apply vectorization to the raybnd function to treat multiple bunches at once
    - Apply the vectorization at the mathematical function level (using dedicated libraries)
    - Want to obtain identical numerical results with respect to a reference version
  - Reduce precision format whenever possible by means of automatic tools

Compiler optimization tests

- Preparatory work
  - Reorganise corsika/sim_telarray packaging (D. Parello)
  - Allowing to easily test different compilation options and code transformations
- Combine different compilation options
  - Standard options:
    - -O1, -O2, -O3
  - Loop optimizations options:
    - -ftree-loop-if-convert -ftree-loop-distribution -ftree-loop-distribute-patterns -ftree-loop-im -ftree-vectorize -funroll-loops -funroll-all-loops -floop-nest-optimize
  - Arithmetics expression optimization (it may affect numerical results):
    - -ffast-math
  - Other options
    - -mavx, -mavx2, -flto

Compiler optimization tests

- Running conditions
  - Same as for profiling
  - Using keep-seeds option for random number generation to obtain reproductible runs
  - Run duration: about 8 minutes
  - Running on a dedicated server
- Performances compared with a reference version compiled with ‘standard’ options
  - -O2 –funroll-loops
- Simple performance measurements with ‘perf stat’: number of cycles, number of instructions, elapsed time, etc.
- Checking result reproducibility
  - Using a dedicated program to print the coordinates of first 10 photons of each bunch

First results of compiler optimizations tests

- 3072 option combinations tested
  - No speed-up obtained beyond a factor 1.06
- Using ffast-math impacts numerical results (as expected)
  - Found that small differences in numerical results may induce different calls to random number generators leading to very different final results

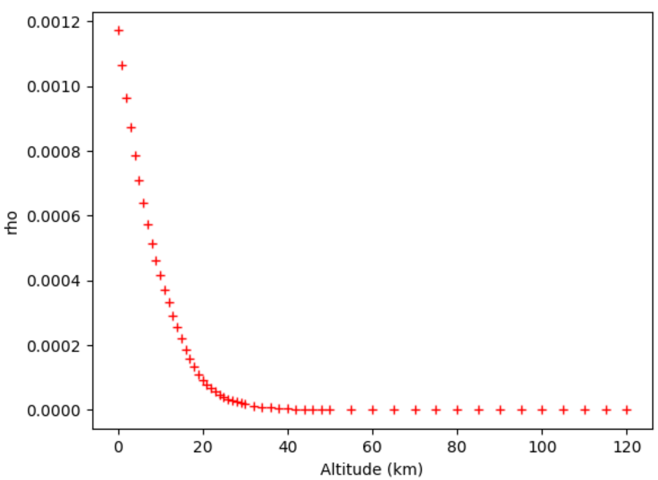
Optimization strategy

- Test automatic optimizations by compiler
  - We don't expect significant gains
- Apply manual transformations
  - At algorithmic level
    - *e.g.* Testing different atmospheric interpolation schemes
  - Code refactoring
  - Exploiting the micro-architecture capabilities
    - Apply vectorization to the raybnd function to treat multiple bunches at once
    - Apply the vectorization at the mathematical function level (using dedicated libraries)
    - Want to obtain identical numerical results with respect to a reference version
  - Reduce precision format whenever possible by means of automatic tools

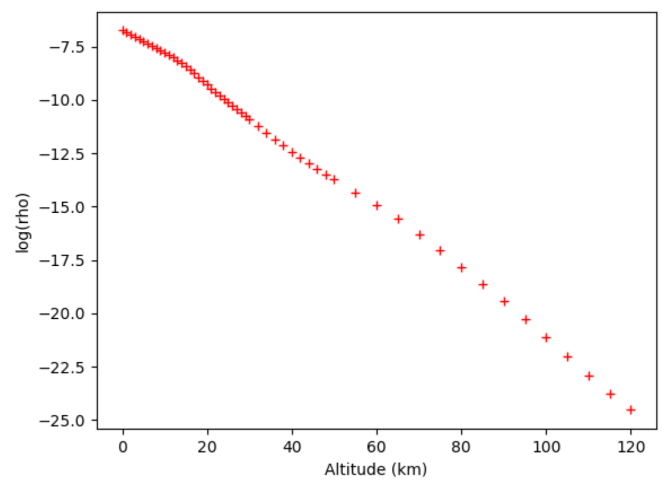
Atmospheric profiles and interpolation

- Generation and propagation of Cherenkov photons require a precise description of the atmosphere in terms of density, thickness, refraction index
- The atmosphere is built from about 55 layers, and then interpolations are used to get precise values at various altitudes
- 35% of CPU time in raybnd spent in computing linear interpolation to evaluate log(density), log(thickness), log(refidx) at various altitudes
  - Implies calls to exp to obtain density, thickness, refraction index values

density profile



log(density) profile



Current interpolation schemes

- Standard interpolation
  - It makes use of binary search algorithm to find the the 2 closest points in the look-up table
- Fast interpolation
  - Enabled by default
  - Use pre-calculated fine-grained tables with equidistant steps in altitude
    - No need anymore of binary search to find the 2 closest points
  - Implemented for atmospheric tables but not for refraction tables

Interpolation schemes

- Comparing the 2 schemes (standard and fast)
  - Fast interpolation gives a speed-up of 1.15
  - Small differences found looking at the corsika output (photon coordinates)
    - x, y at micron level
    - Arrival time at < 0.1 ps level
    - No angular differences
- Started the extension of fast interpolation to refraction tables
  - No significant gain for the moment (though very preliminary)
- We've confirmed that interpolation algorithm has an impact on performances
- Other algorithms may be implemented in future (quadratic, cubic-splines)
  - Will allow to avoid exp calls
  - Accuracy of interpolation results need to be carefully checked

Optimization strategy

- Test automatic optimizations by compiler
  - We don't expect significant gains
- Apply manual transformations
  - At algorithmic level
    - e.g. Testing different atmospheric interpolation schemes
    - Numerical results may be slightly different (need be carefully validated)
  - Code refactoring
  - Exploiting the micro-architecture capabilities
    - Apply vectorization to the raybnd function to treat multiple bunches at once
    - Apply the vectorization at the mathematical function level (using dedicated libraries)
    - Want to obtain identical numerical results with respect to a reference version
  - Reduce precision format whenever possible by means of automatic tools

First manual optimization

- In raybnd function (by DP v_opt001)
- Observation of redundant calls to ‘binary search’ function for atmospheric and refraction tables interpolation
- Simple code transformation to eliminate redundant calls
  - Speed-up of 1.09
  - No differences in final bunches coordinates
  - Bonus
    - Expose vectorization possibilities for exp calls

Second manual optimization

- Using a library vectorizing the most common mathematical functions (exp, log, sin, cos, etc.) v_opt002
  - https://hal.archives-ouvertes.fr/hal-01511131/document
  - Announced speed-up of 280% for exp
- Starting from version v_opt001
  - Replace in raybnd 3 exp calls to 1 vector exp call

```
*rhofx = exp(p_log_rho[ipl-1]*(1.-rpl) + p_log_rho[ipl]*rpl);
*thickx = exp(p_log_thick[ipl-1]*(1.-rpl) + p_log_thick[ipl]*rpl);
*refidx = 1.+exp(p_log_n1[ipl-1]*(1.-rpl) + p_log_n1[ipl]*rpl);
```

- Speed-up of 1.16
- No differences in final bunches coordinates
- Similar results obtained with vector exp developed by G. Revy
  - Version with simple precision

Start implementing vectorization

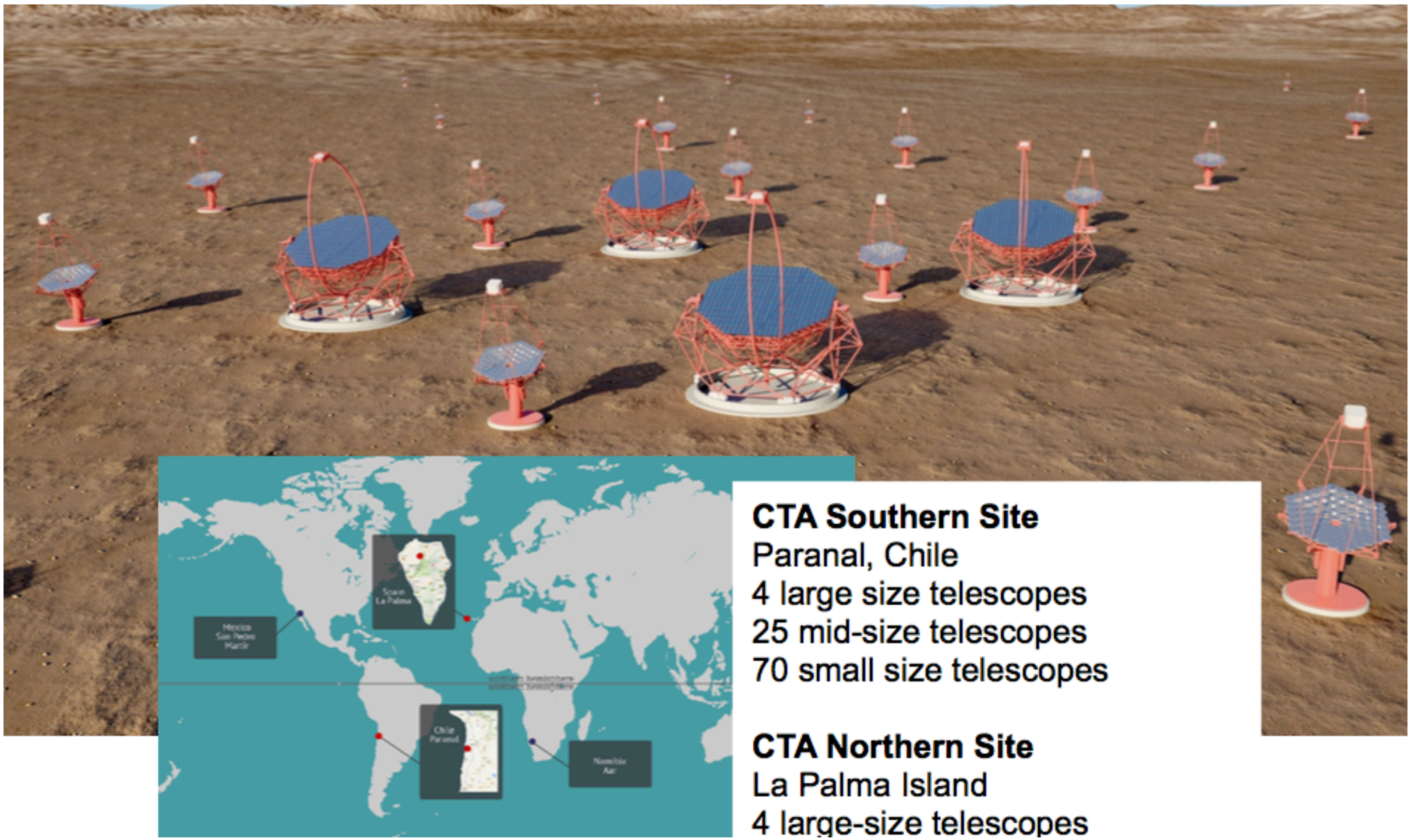
- Testing different libraries for an easier vectorization on different architectures
  - bSIMD
    - https://developer.numscale.com/bsimd/documentation/v1.17.6.0/
  - UME (Unified Multicore Environment)
    - https://gain-performance.com/ume/
- Both require C++ compiler and don't support vectorized mathematical functions
- First attempt vectorizing 'binary search' function using UME
  - Atmospheric tables are relatively small (*e.g.* 55 points)
  - Avoid binary search and simply group table elements by 4 or 8 to perform comparisons with the searched value
  - No significant speed-up observed (using a different algorithm though)

Conclusions

- Preliminary work started for corsika optimization in collaboration with computer scientists (LIRMM/UPVD)
- Focusing on photon propagation in the atmosphere
- 1.16 speed-up already obtained with simple code transformation and limited application of vectorized mathematical libraries (with the constraint of getting identical results wrt reference version)
- Next steps
  - Extend the vectorization in raybnd to other calculations
  - Start the work on precision reduction
- Workshop at KIT next week about the corsika re-writing project
- The goal is to integrate the optimizations in the new software framework

BACKUP

CTA


CTA Southern Site
Paranal, Chile
4 large size telescopes
25 mid-size telescopes
70 small size telescopes
CTA Northern Site
La Palma Island
4 large-size telescopes

corsika packages

fully 4-dim.

tracking, decays, atmospheres, ...

* recommended
* based on Gribov-Regge theory
* source of systematic uncertainty

el.mag.	EGS4 *
low-E.had.*	FLUKA *
	UrQMD
	GHEISHA
high-E.had. **	QGSJET **
	EPOS-LHC *
	DPMJET *
	SIBYLL

Tuned at collider energies, extrapolated to > 10^{20} eV

+ many extensions & simplifications

Sizes and runtimes vary
by factors 2 - 40.

Total: >> 10^5 lines of code

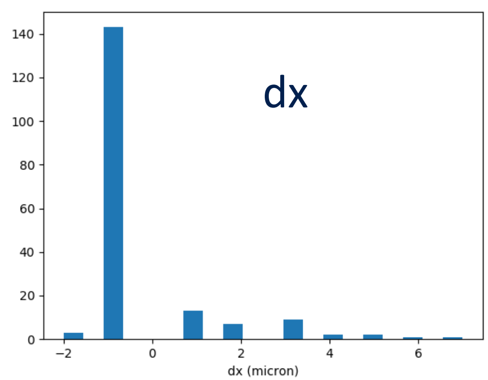
many person-years

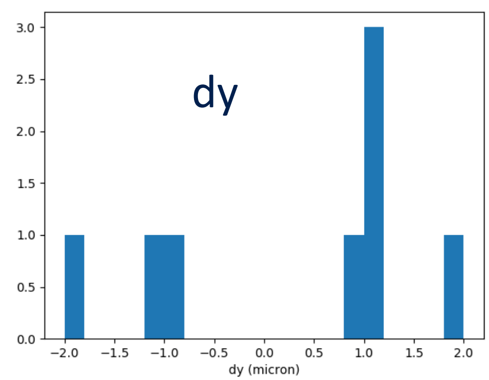
Interpolation in raybnd

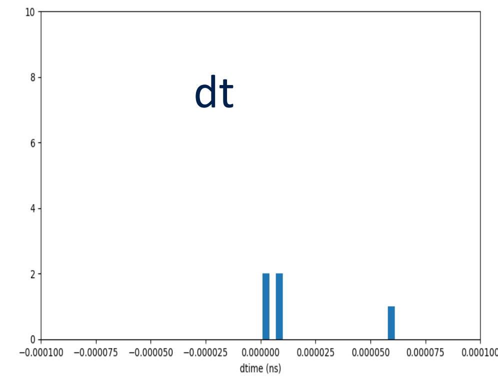
- In raybnd (for non vertical paths)
  - 3 fast interpolations (calls to thickx_, refidx_, rhofx_)
    - Interpolation of atmospheric tables
    - Evaluate thickness, refraction index and density at the emission altitude
    - Also other calls directly from cerenk
  - 3 standard interpolations with binary search (calls to rpol)
    - Interpolation of refraction tables
    - Evaluate horizontal displacement and time offset for a given density or altitude
    - Fast Interpolation not implemented for refraction tables
- Comparing the 2 schemes (standard and fast)
  - Fast interpolation gives a speed-up of 1.15
  - Small differences found looking at the corsika output (see next slide)
  - Started the extension of fast interpolation to other tables but no significant gain obtained for the moment

Interpolation schemes

- Small differences found in bunch coordinates (standard vs fast interpolation)
  - x, y at micron level
  - arrival time at < 0.1 ps level
  - no angular differences







- Problem of the validation of new code versions
  - Benchmark definition
  - Acceptable deviations from reference version