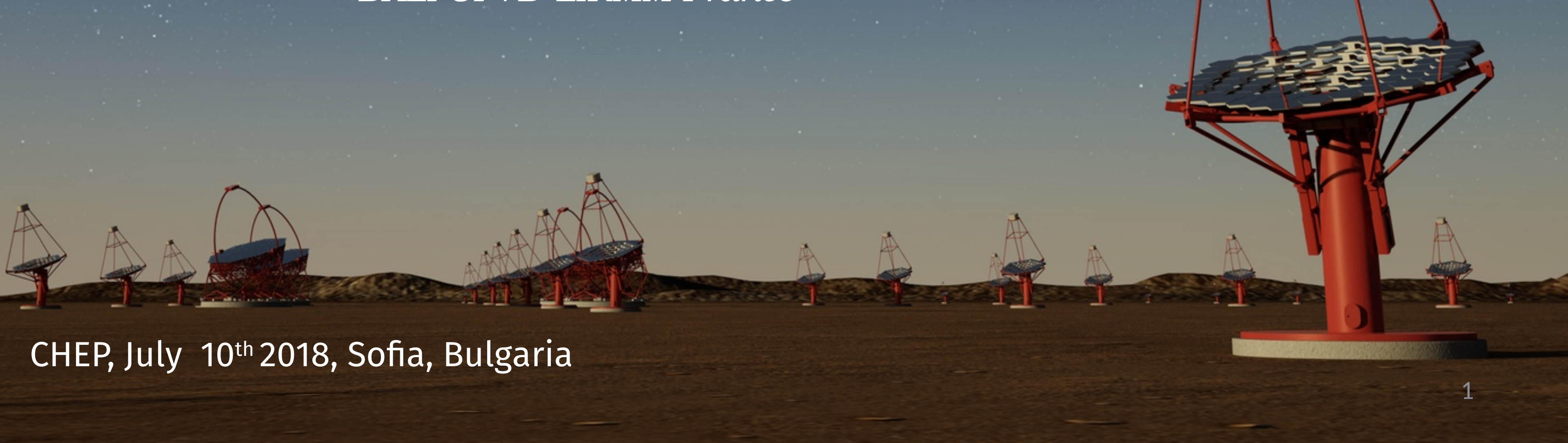
Preliminary work on optimizing the Corsika air shower simulation program for CTA

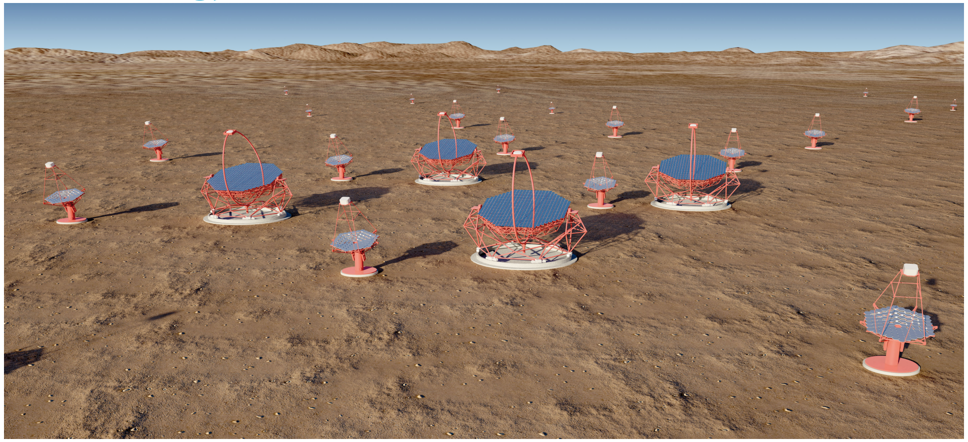
L. Arrabito[1], J. Bregeon[1],
P. Langlois[2], D. Parello[2], G. Revy[2]
[1]LUPM CNRS-IN2P3 France
[2]DALI UPVD-LIRMM France



CHEP, July 10th 2018, Sofia, Bulgaria

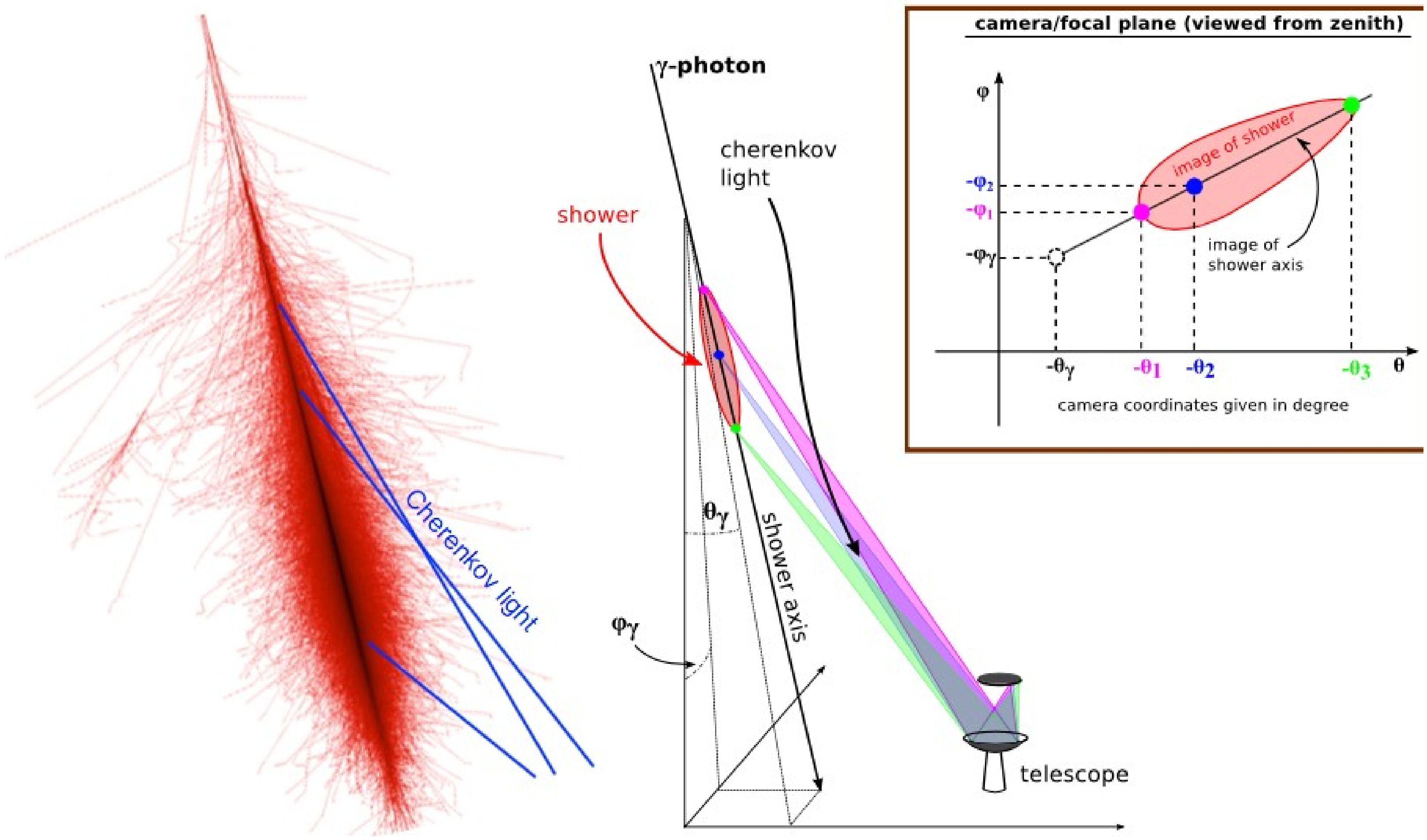
CTA project

- The next generation instrument in VHE gamma-ray astronomy (1400 scientists in 31 countries)
  - Cosmic ray origins, High Energy astrophysical phenomena, fundamental physics and cosmology



- Two arrays of Cherenkov telescopes
  - Northern hemisphere (La Palma, Spain): 4 LSTs, 15 MSTs
  - Southern hemisphere (Paranal, Chile): 4 LSTs, 25 MSTs, 70 SSTs
- Project schedule
  - Construction and deployment: 2019-2025
  - Science operations: start in 2022 for 30 years

Imaging Atmospheric Cherenkov Telescope



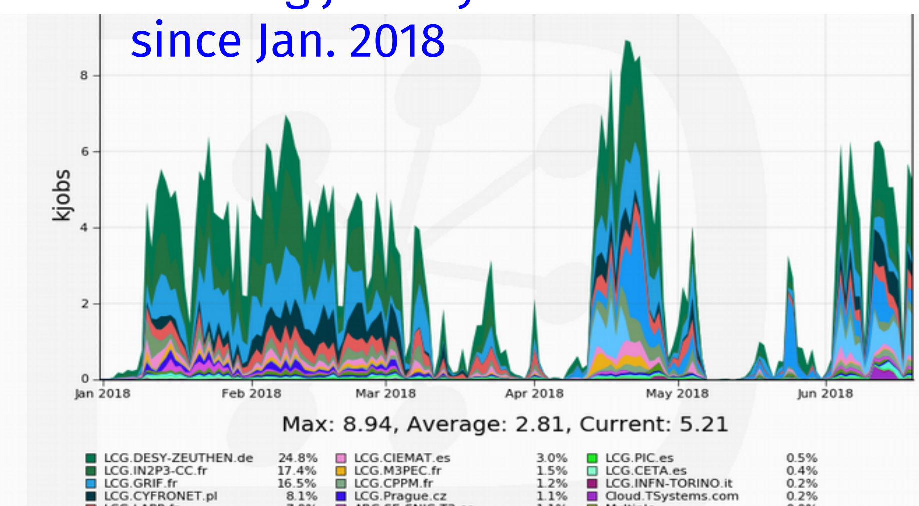
Corsika Air shower simulation

- Detailed simulation of showers initiated by high energy cosmic rays
  - 30 years old Fortran program of more than 10^5 lines of code
  - Initially developed for the Kaskade experiment (since 1990 at the Karlsruhe Institute for Technology)
  - Widely used by several ‘cosmic rays’ communities (Veritas, Auger, JEM-EUSO, IceCube…)
  - 900 users from 57 countries and > 1900 citations
- Customized external packages for electromagnetic and hadron interactions (mostly Fortran)
  - EGS4, FLUKA, UrQMD, GHEISHA, QGSJET, EPOS-LHC, DPMJET, SIBYLL
- IACT/atmo package (written in C, the “Bernloehr” package)
  - Extension to Corsika to implement arrays of Cherenkov telescopes
  - Use of external atmospheric models
  - Propagation of the Cherenkov light in the atmosphere with refraction

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



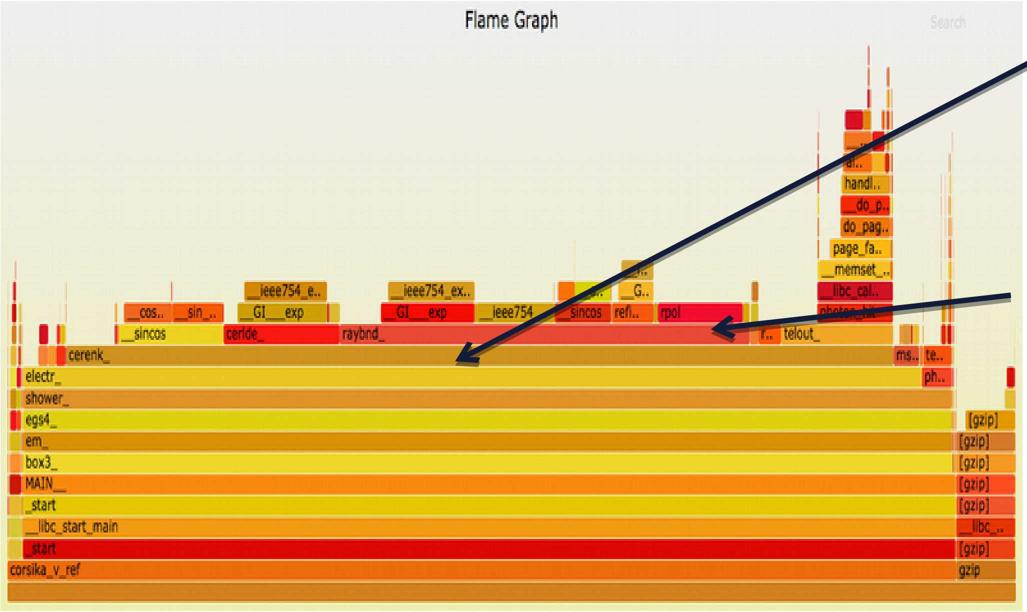
8000 jobs

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

See L. Arrabito's talk Thursday on CTADIRAC production setup.

Corsika "CTA production setup" profiling with Linux perf

Linux perf + FlameGraph

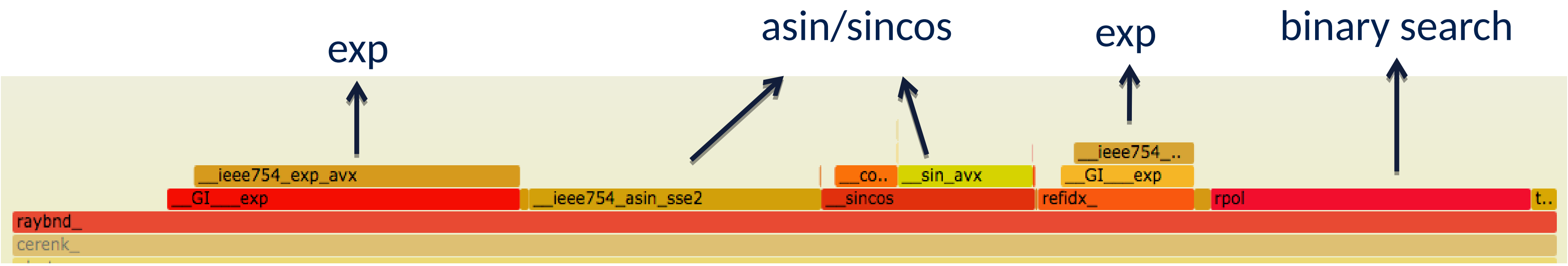


- 90% of CPU in *CERENK* subroutine and below
  - Cherenkov photon production
  - Part of Corsika 'core'
- 50% of CPU in *raybnd* function and below
  - Propagation of Cherenkov photon in the atmosphere with refraction correction
  - Part of IACT/atmo package

- 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

Reference setup

- Dedicated server
  - Intel(R) Xeon(R) CPU E5-2650 v4 @ 2.20GHz running CentOS Linux 7.4 - 64 bits
- Running conditions
  - Standard “CTA Production setup” - same as for profiling
    - compiled with standard options “-O2 -funroll-loops”
  - Using **keep-seeds** option for random number generation to obtain reproducible runs
  - Run duration: about 8 minutes
- Simple performance measurements with ‘**perf stat**’: number of cycles, number of instructions, elapsed time, etc.
- Checking result reproducibility
  - Goal to obtain identical numerical results with respect to a reference version
  - Using a dedicated program to print the coordinates of first 10 photons of each bunch
  - ➔ Need to develop better tools!

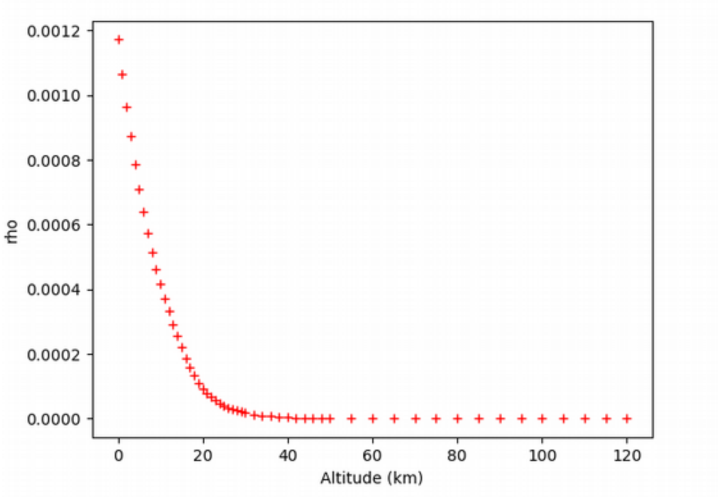
Optimization strategy

- Test automatic optimization by compiler
  - 3072 options combinations… but no significant improvement (as expected)
  - "*-ffastmath*" in particular does not bring any significant improvement and slightly different numerical results
- 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
  - Reducing precision format whenever possible by means of automatic tools

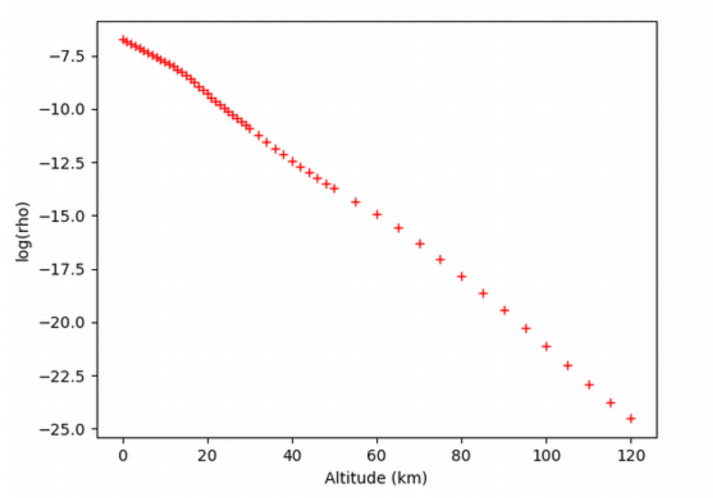
Atmospheric profiles and interpolation

- Generation and propagation of Cherenkov photons require a precise description of the atmosphere: **density, thickness, refraction index**
- The atmosphere is built from 55 layers, and then interpolations are used to get precise values at various altitudes
  - Find the 2 closest points in the table and then compute a linear interpolation
- 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
  - Efficient algorithm can be run on any table, but it runs many times
- **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 density and thickness 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
- We've confirmed that interpolation algorithm has an impact on performances
- Started the extension of fast interpolation to refraction tables
  - No significant gain for the moment (though very preliminary)
- 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
  - Carefully check the consistency of interpolations between the 3 tables

Manual optimization

- Refactoring the *raybnd* function
  - Redundant calls to 'binary search' function for atmospheric and refraction tables interpolation
  - Simple refactoring to eliminate redundant calls
    - Speed-up of 1.09 for identical numerical results
    - Bonus : vectorization possibilities for exp calls

```
*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);
```

- Using a library vectorizing the most common mathematical functions (exp, log, sin, cos, etc.)
  - Announced speed-up of 280% for exp
  - In *raybnd*, **replaced 3 exp calls to 1 vector exp call**
    - Speed-up of **1.16** for identical numerical results
- Similar results obtained using a local custom "simple precision" version of the library developed by G. Revy

Start implementing vectorization

- Testing different libraries for an easier vectorization of simple mathematical operations 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…
    - not applicable to Corsika core
  - and are not compatible with our preferred library of vectorized mathematical functions
- First attempt vectorizing ‘binary search’ function **using UME**
  - Atmospheric tables are relatively small (*e.g.* 55 points)
  - Replace binary search with brute force vectorized algorithm
    - group table elements by 4 or 8 in vectors and perform comparisons with the searched value for all elements in one call

  ➔ not faster than binary search

Start implementing vectorization

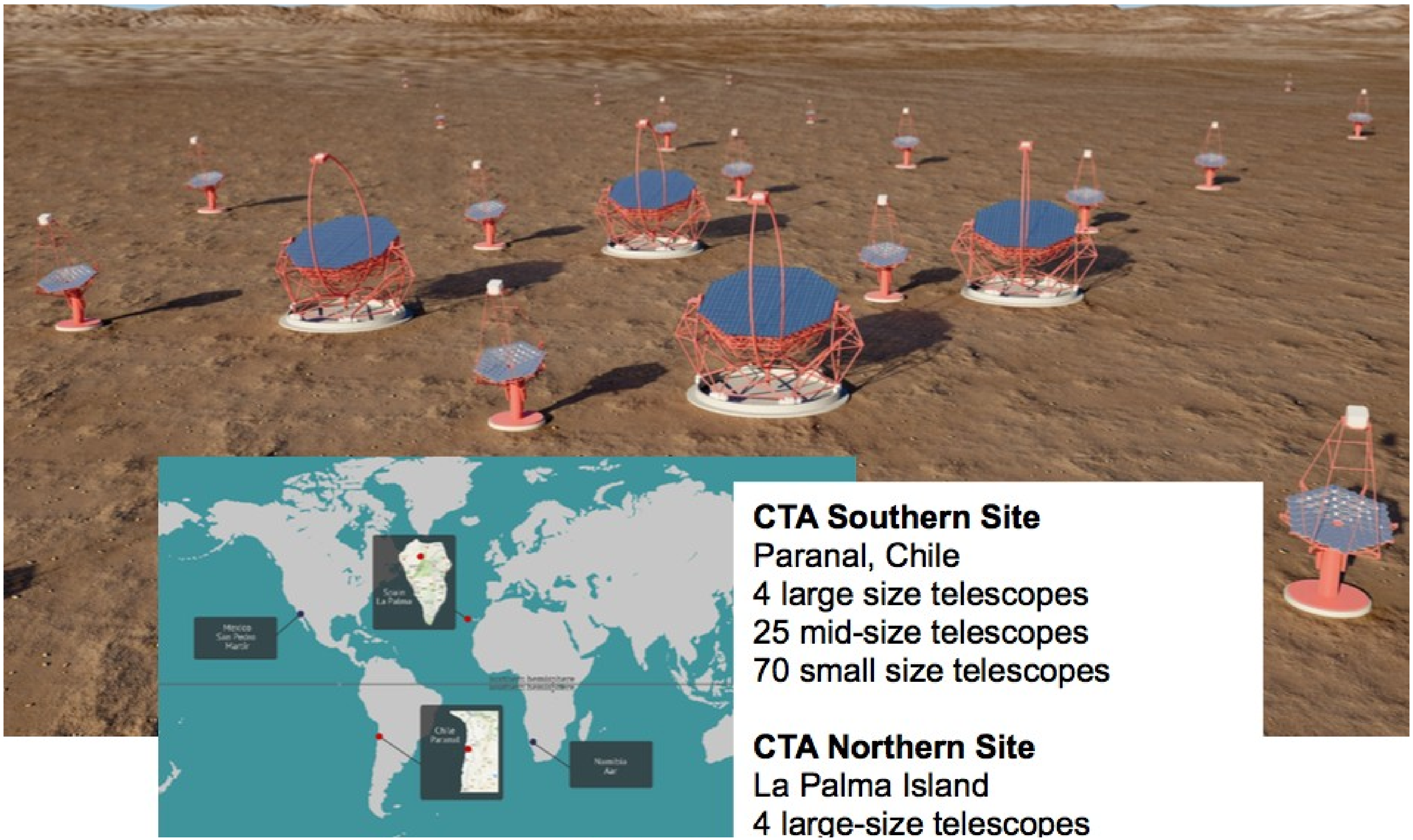
- Start vectorizing *raybnd* function
  - Had to modify the loop calling *raybnd* function to pass vector arguments
    - Unroll loop to process consecutively 4 photon bunches
  - In *raybnd*: replace all calls to asin, sin, cos with their vector counterparts (using a dedicated library)
    - Speed-up: 1.11
- Combining this optimization with ‘code refactoring + vector_exp’ optimization (slide 13)
  - **Speed-up: 1.28**

Conclusions

- Preliminary work started for Corsika optimization in collaboration with computer scientists (LIRMM/UPVD)
- Focusing on photon propagation in the atmosphere
- **1.28 speed-up** already obtained with simple code transformation and limited application of vectorized mathematical libraries
  - with the constraint of getting identical results w.r.t. reference version
- Next steps
  - Extend the vectorization in *raybnd* to other calculations
  - Start the work on precision reduction
  - Develop automatic optimization tools
- Workshop at KIT on June 25th-26th about the **New Generation Corsika** project - C++!
  - https://indico.scc.kit.edu/event/426/
  - Work also in the new framework on the mid-term

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, ...

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

+ many extensions & simplifications

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

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

Sizes and runtimes vary by factors 2 - 40.

Total: >> 10^5 lines of code

many person-years

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)

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

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

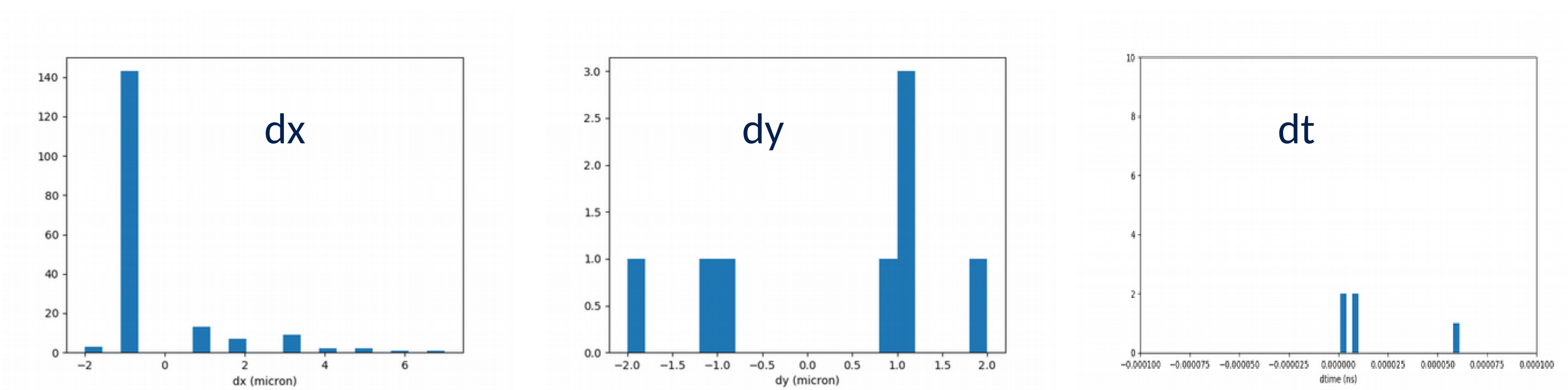
Interpolation in raybnd

- 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