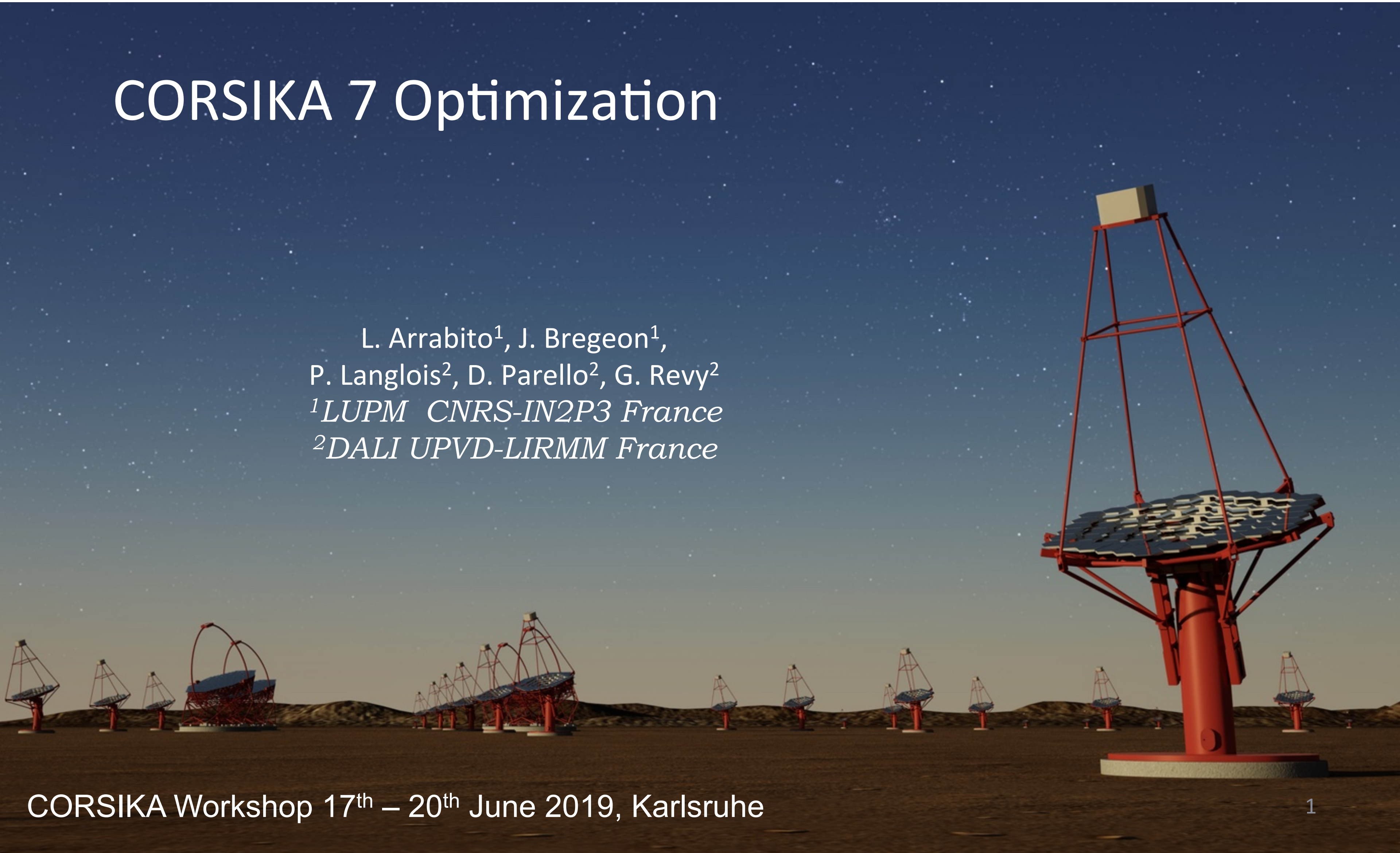
CORSIKA 7 Optimization

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*



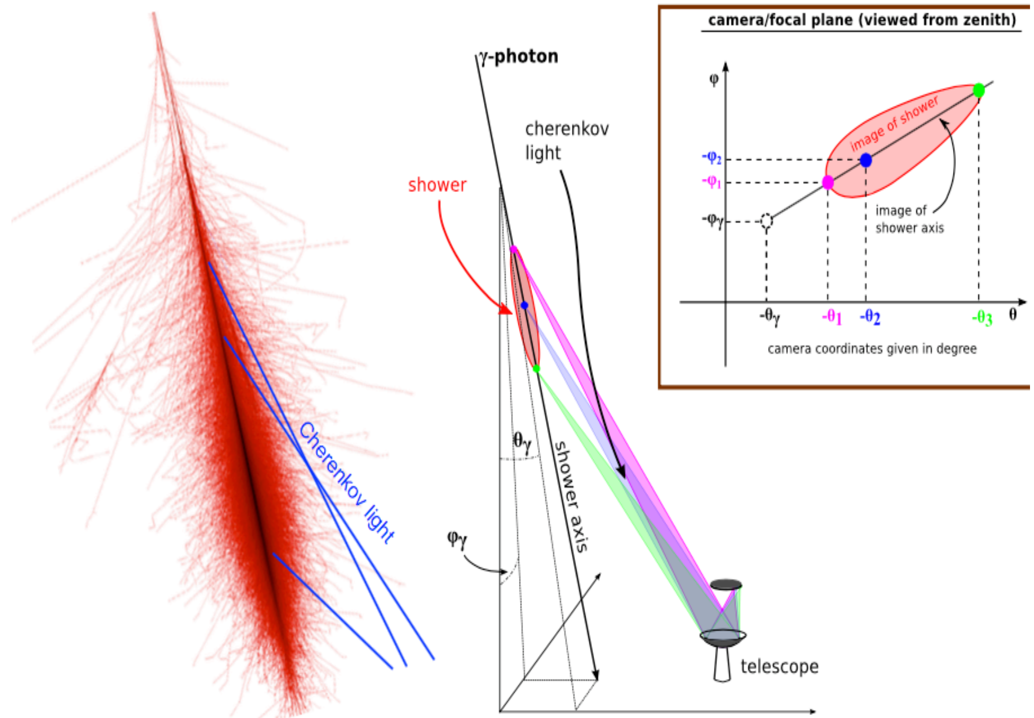
CORSIKA Workshop 17th – 20th June 2019, Karlsruhe

Plan

- Motivations for CORSIKA 7 Optimization
- Profiling
- Vectorization of the Cherenkov module
- Performance results
- Conclusions and plans

CORSIKA for CTA

- Detailed simulation of showers initiated by high energy cosmic rays
- Customized external packages for electromagnetic and hadron interactions (mostly Fortran)
- IACT/atmo package (written in C)
  - Extension to CORSIKA to implement arrays of Cherenkov telescopes
  - Use of external atmospheric models
  - Propagation of Cherenkov light in the atmosphere with refraction

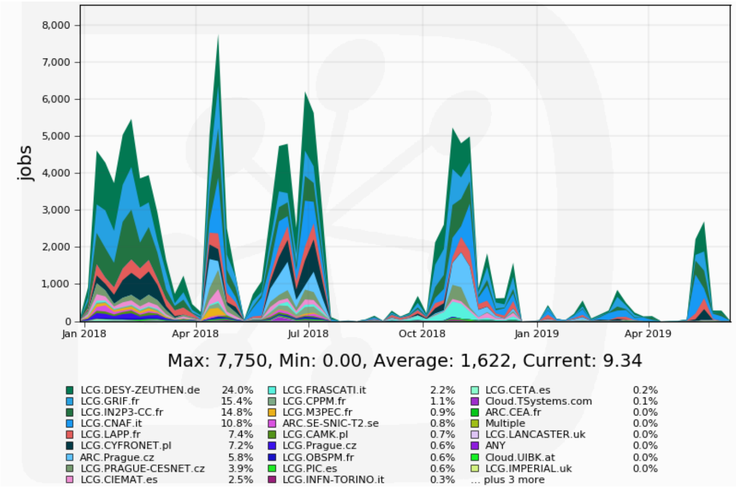


Motivations for CORSIKA optimization

- MC simulations in CTA are the most CPU consuming task
  - 70% in CORSIKA and 30% in telescope simulation
- Massive MC simulations run on the grid since 2012 to assess CTA design
  - Consuming 100-200 M HS06 CPU hours/year
- High CPU requirements are expected also during CTA operations
- Even a small speed-up will save millions of CPU hours

Running jobs by site since Jan. 2018

8000 jobs



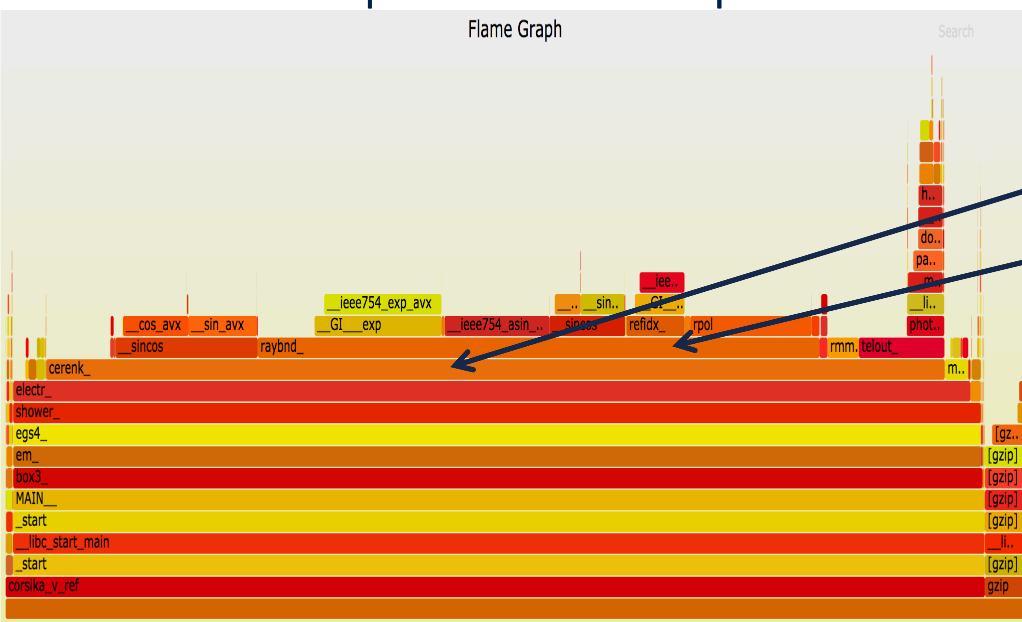
Reference setup

- Dedicated server
  - x86_64 Intel(R) Xeon(R) CPU E5-2650 v4 @ 2.20GHz running CentOS Linux release 7.4 - 64 bits
- Running with standard CTA production parameters
  - qgs2 interaction model
  - PRIMARY gamma point source
  - THETAP 20 and PHIP 180
  - ERANGE 3.0 330E3 and ESLOPE -2.0
  - CSCAT 10 2000e2 0.
  - External Atmosphere
  - Using **keep-seeds** option for random number generation to obtain reproductible runs
  - 1000-5000 showers run

Profiling CORSIKA 6/IACT 1.51

- Work started in early 2018 with CORSIKA 6/IACT 1.51
- Profiling for CTA ‘standard’ running conditions

Linux perf + FlameGraph

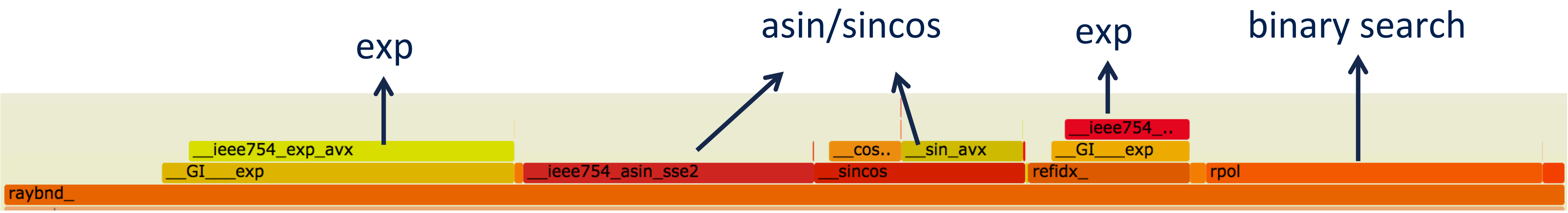


- 88% CPU in *CERENK*
- 55% *raybnd* (IACT/atmo)
- 14% sincos
- 8% telout
- ...
- Note: LONGI disabled

- IACT 1.51
  - ‘Old’ atmospheric interpolation scheme (see slide 7)

Profiling CORSIKA 6/IACT 1.51

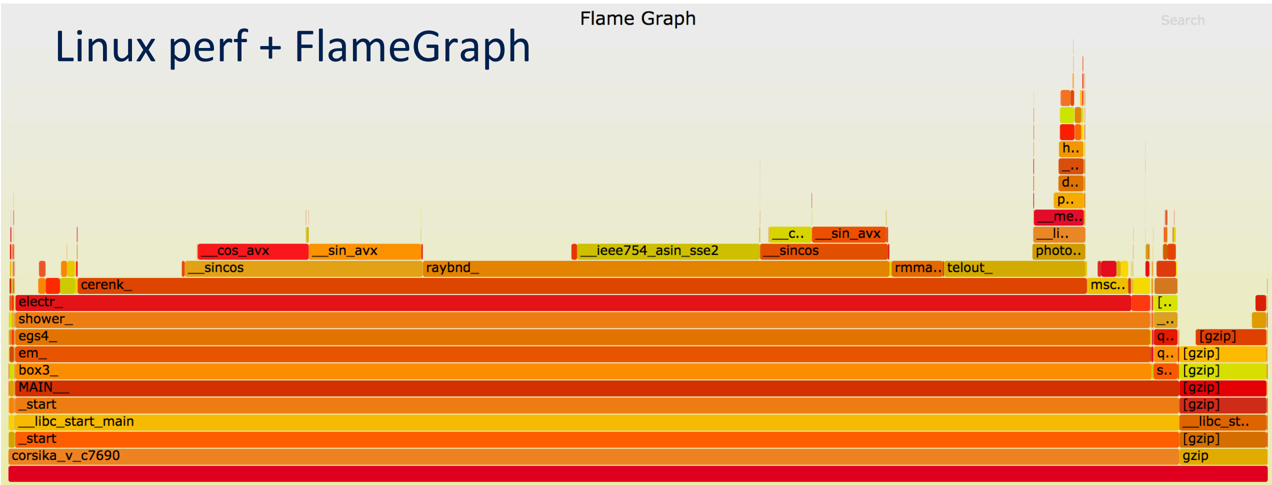
- Zoom on *raybnd*



- Most of the CPU spent in **mathematical functions** and atmospheric/refraction **profile interpolation**
  - 32% exp (used for atmospheric profile interpolation)
  - 33% sincos/asin
  - 21% atmospheric interpolation with binary search
- Very frequently called function, once per photon bunch
- Computations on photon bunch propagation are independent
  - **Good candidates for vectorization**

➡ **Choose to start optimizing raybnd/CERENK using vectorization techniques**

Profiling CORSIKA 7/IACT 1.59

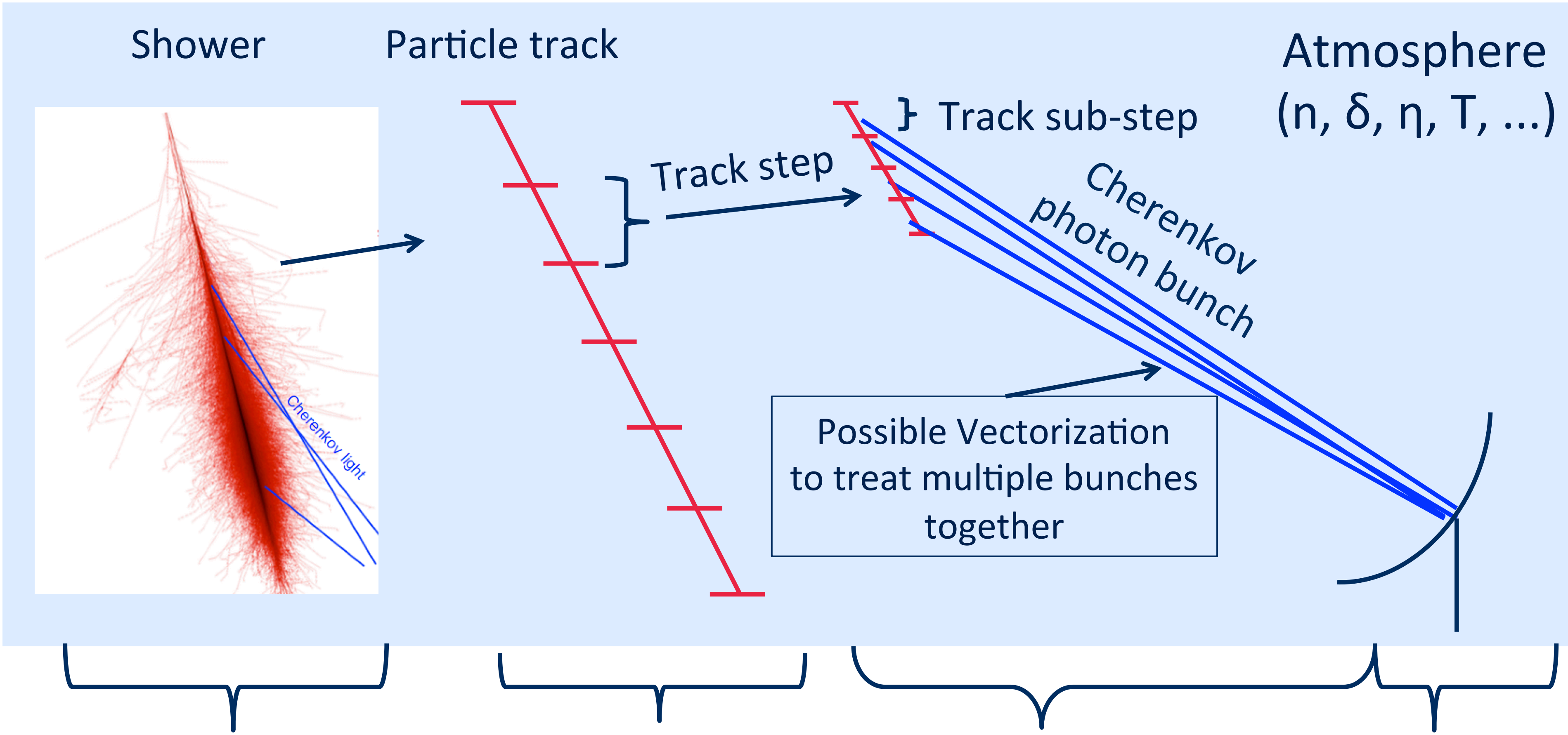


- CERENK: 80%
- raybnd: 37%
- sincos: 19%
- telout: 11%
- rmmard: 4%
- Note: LONGI disabled

Zoom on raybnd

- IACT 1.59
  - New faster interpolation scheme (by Konrad Bernlohr)
    - Using fast interpolation throughout (no binary search anymore)
    - Avoiding exp calls in the interpolation process
  - In raybnd: 67% CPU in asin/sincos

Cherenkov production and propagation

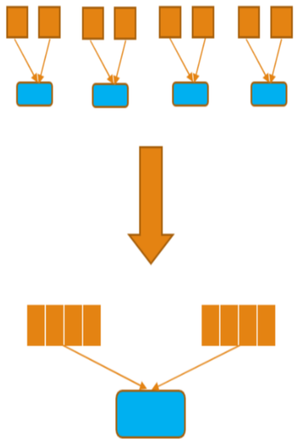


EGS4/SHOWER/ELECTR	CERENK	raybnd	telout
- Shower development - Transport of electron/positron	- Nb of emitted Cherenkov photons - Grouped in bunches	- Deviation of photon bunches by atmospheric refraction	- Record coordinates of photon bunches hitting the telescopes

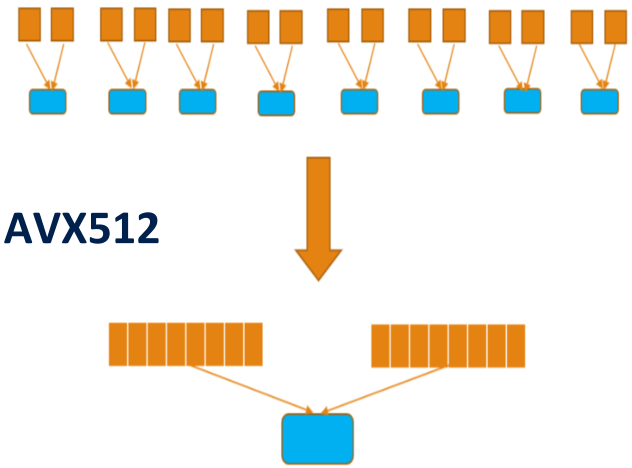
Vectorization

- Variables stored in vector registers
- SIMD (Single Instruction on Multiple Data) instructions
  - Perform the same operation on multiple variables in parallel
- Most common SIMD instructions
  - +, -, *, /, mask, horizontal operations etc.
- AVX2
  - Registers of 256 bits
    - 4 doubles or 8 floats
  - Commonly available in computing centers (*e.g.* grid sites)
- AVX512
  - Registers of 512 bits
    - 8 doubles or 16 floats

AVX2



AVX512



Vectorization techniques

- Explicit calls to low level SIMD instructions (intrinsics)
  – Complex syntax
  – Architecture dependent
- Using vector libraries
  – Provide an abstraction of low level SIMD instructions
  – Portable on different architectures
- Auto-vectorization
  – The compiler automatically detects vectorizable patterns and perform the vectorization
  – It works with ‘simple patterns’
  – Enabled by compiler flags: *e.g.* -03, -mavx2

Vector libraries

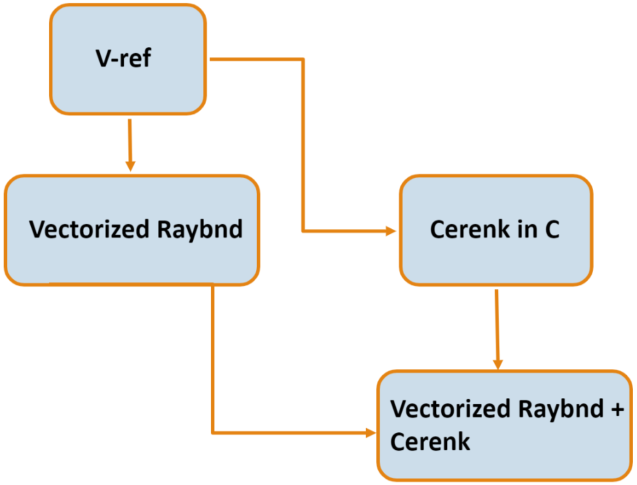
- **Provide an abstraction of low level SIMD instructions (intrinsics)**
  - **Allow to transparently vectorize arithmetics expression (+, -, /, ∗) on different architectures**
- Vc
  - https://github.com/VcDevel/Vc
  - SSE4, AVX, AVX-2
- UME::SIMD **-> Tested**
  - https://github.com/edanor/umesimd
  - SSE4, AVX, AVX-2, AVX-512
  - Support of some vectorized mathematical functions but not really optimized
- bSIMD **-> Tested**
  - https://developer.numscale.com/bsimd/documentation/v1.17.6.0/
  - Only partially open source
  - SSE4, AVX, AVX-2, AVX-512
- xsimd
  - https://github.com/QuantStack/xsimd
- VecCore (CERN project)
  - https://github.com/root-project/veccore
  - Uses Vc and UME::SIMD as backend

Vectorized math libraries

- **Implement vectorized version of mathematical functions**
  - **exp, log, sin, cos, acos, asin, sqrt, cbrt, ...**
  - **They are less accurate than standard scalar libm but enough for many applications**
- Intel's SVML
  - https://software.intel.com/en-us/node/583201
- AMDs libm
  - http://developer.amd.com/tools-and-sdks/archive/libm
- GNU's libmvec (open source) **-> Tested**
  - gcc > 4.9.0
  - glibc > 2.26
  - Enabled by ffast-math flag
  - https://sourceware.org/glibc/wiki/
- CERN's VDT (backend of VecCore) (open source) **-> Tested**
  - https://github.com/dpiparo/vdt
  - http://iopscience.iop.org/article/10.1088/1742-6596/513/5/052027/pdf
- SIMD vector libm (open source) **-> Tested**
  - https://gitlab.com/cquirin/vector-libm
  - https://hal.archives-ouvertes.fr/hal-01511131/document

CORSIKA 7 optimization strategy 1/2

1. Test automatic optimizations by compiler
   - No change in the code
   - Extensive tests done with several gcc flags -> No significant gain
2. Manual code transformation
   - Vectorize as much as possible: raybnd, CERENK (and CERLDE)
   - CERENK re-written in C for an easier code transformation
   - ‘Incremental’ transformations
   - Evaluate the speed-up of each transformation



CORSIKA 7 optimization strategy 2/2

- Choose the auto-vectorization approach
  - No use of machine dependent SIMD instructions (intrinsics)
  - No external dependencies on vector libraries
  - **It needs to transform the code to allow the compiler to detect vectorizable patterns**
- Use of the SIMD vector libm only for vectorizing mathematical functions (exp, cos, sin, etc.)
  - https://gitlab.com/cquirin/vector-libm

Vectorization of raybnd, CERENK, CERLDE 1/3

- Code transformations
  - Unroll the sub-step loop in CERENK containing the call to raybnd
    - Allows to pass 4-lenght vectors to raybnd instead of scalars

```
// VECTORIZED
for( int i=0; i<VECTOR_SIZE; i++)
  cartim_vec[i] = temis_vec[i] * 1e9;
raybnd_vec_(zem_vec, uemis_vec, vemis_vec, wemis_vec, xcer_vec, ycer_vec, cartim_vec);
  }
```

  - Restructure the 'if' tests
  - Isolate computations to facilitate the detection of vectorizable loops by compiler
  - Inlining CERLDE
- Compiler flags: -O3, -mavx2
- Check that vectorization is effective by looking at the assembler code

Vectorization of raybnd, CERENK, CERLDE 2/3

- Example of transformation in raybnd

Original version

```
*u *= sin_t_obs/sin_t_em;
*v *= sin_t_obs/sin_t_em;

if ( (*w) >= 0. )
   *w = sqrt(1.-sin_t_obs*sin_t_obs);
else
   *w = -sqrt(1.-sin_t_obs*sin_t_obs);

*dx += hori_off * (*u)/sin_t_obs;
*dy += hori_off * (*v)/sin_t_obs;
*dt += travel_time;
```

Optimized version

```
// VECTORIZED
for(int i=0; i< VECTOR_SIZE; i++){
  u[i] *= sin_t_obs[i]/sin_t_em[i];
}
// VECTORIZED
for(int i=0; i< VECTOR_SIZE; i++){
  v[i] *= sin_t_obs[i]/sin_t_em[i];
}
// VECTORIZED
for(int i=0; i< VECTOR_SIZE; i++){
  w[i] = sqrt(1.-sin_t_obs[i]*sin_t_obs[i]);
}
// VECTORIZED
for(int i=0; i< VECTOR_SIZE; i++){
  dx[i] += hori_off[i] * (u[i])/sin_t_obs[i];
}

// VECTORIZED
for(int i=0; i< VECTOR_SIZE; i++){
  dy[i] += hori_off[i] * (v[i])/sin_t_obs[i];
}

// VECTORIZED
for(int i=0; i< VECTOR_SIZE; i++){
  dt[i] += travel_time[i];
}
```

Vectorization of raybnd, CERENK, CERLDE 3/3

- Example of transformation in raybnd
  - Using the SIMD vector libm for vectorizing mathematical functions, *e.g.* sin, cos, etc.

Optimized version

Original version

```
c = cos(theta_em+0.28*(theta_obs-theta_em));
s = sin(theta_em+0.28*(theta_obs-theta_em));
```

```
// VECTORIZED
for (int i=0; i < VECTOR_SIZE; i++){
  c_x[i] = theta_em[i]+0.28*(theta_obs[i]-theta_em[i]);
}

vector_cos(c,c_x);
vector_sin(s,c_x);
```

Performance measurements and validation

- Performance measurements with ‘**perf stat**’ (*e.g.* number of cycles, elapsed time, etc.)
- Speed-up = *ExecutionTime(ref_ver)/ExecutionTime(opt_ver)*
- Validation
  - Read photon bunches coordinates in the CORSIKA output
    - x, y, cosx, cosy, arrival time
  - Compare with the reference version using
    - *read_cta* program (by Konrad Bernlhor)
    - Python library: https://github.com/cta-observatory/pyeventio

Performance results

- Final results obtained with
  - Compiler: **gcc 8.2.1**
  - Compilation flags: **-O3 -mavx2**
  - CentOS 7 (with AVX2)
- Speed-up
  - raybnd vectorized only: 1.21
  - raybnd+cerenk vectorized: 1.52
  - raybnd+cerenk vectorized on AVX 512: 1.60
- Optimized versions validated against the reference version

Report on CTAOptSim Workshop

- ‘CTAOptSim’ Workshop (supported by CNRS as ‘Astro-Informatique’ project) held in Montpellier, December 2018
  - https://gite.lirmm.fr/cta-optimization-group/cta-optimization-project/wikis/lupm_december_2018
- Focus on optimization of MC simulation codes
  - Collect ideas to design CORSIKA 8 with optimization in mind
- Talks on:
  - CORSIKA 8 and CORSIKA application in CTA
  - GEANT4, GeantV
  - Vectorization and math libraries
  - Numerical accuracy
- Bringing together physicists, simulation experts and computer scientists (from CERN, KIT, CTA Consortium and University of Perpignan)

Conclusions and plans

- Good progress made on CORSIKA 7 optimization using auto-vectorization in Cherenkov module
  - Speed-up 1.52
- A few jobs run on the grid to check the portability and the speed-up results -> To be done at a larger scale
- Future plans for optimization
  - Memory access patterns
  - Eventual precision reduction in some parts of the code
- PhD on CORSIKA 7 optimization and contribution to CORSIKA 8 project will start in October 2019 at LUPM and University of Perpignan