DALI
Digits Architectures Logiciels Informatique

cnrs

UNIVERSITÉ PERPIGNAN VIA DOMITIA

LIRMM

Optimizing Cherenkov photons generation and propagation in Corsika

ADNANE KHATTABI

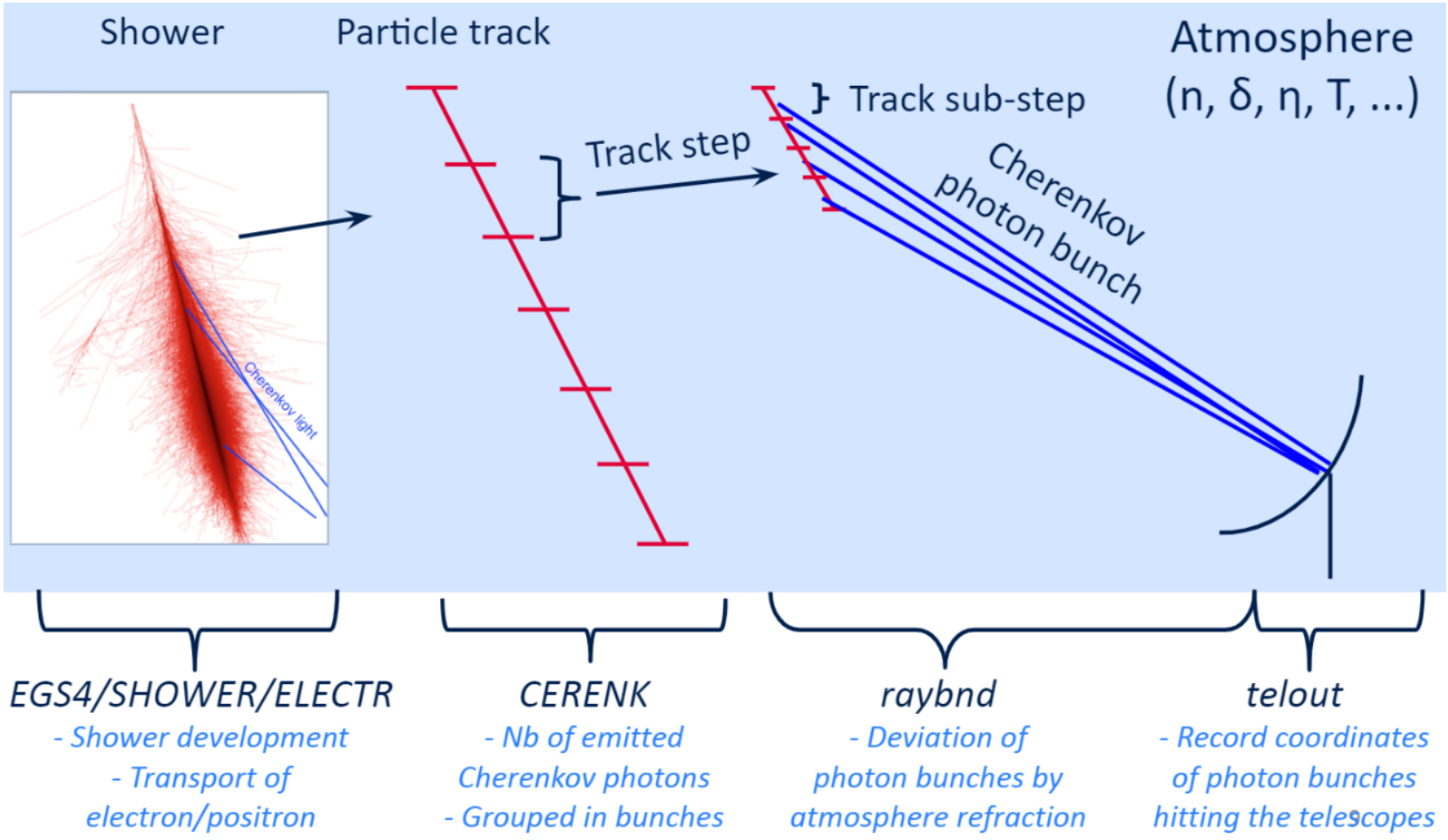
LUPM
LABORATOIRE MONTPELLIER
UNIVERS & PARTICULES

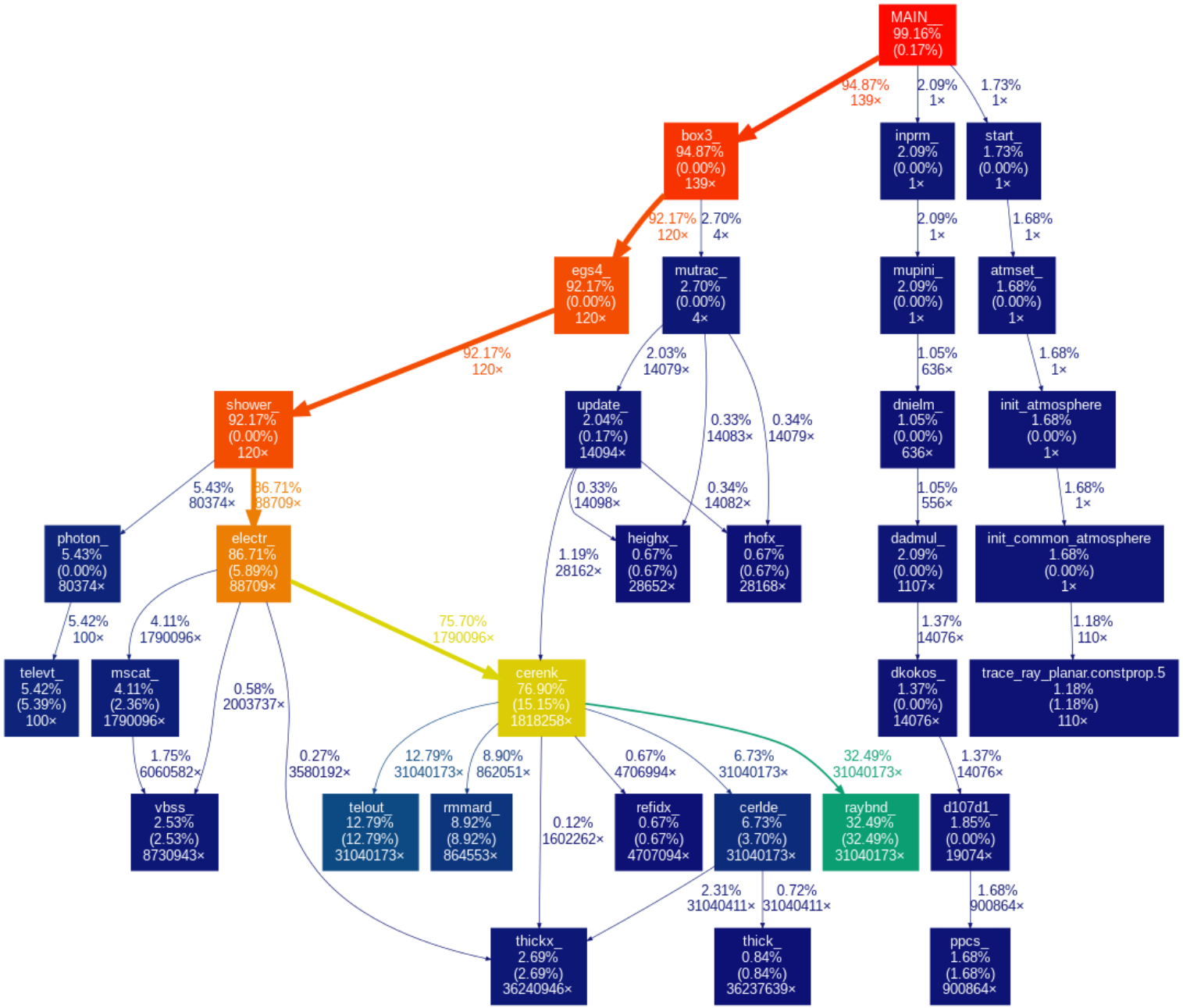
cta
cherenkov telescope array

Content

1. Cherenkov photons production and propagation.
2. Corsika profiling.
3. Optimization process.
    1. Raybnd
    2. Cerenk
    3. Cerlde
4. Experiments and performance.
5. Conclusion

Cherenkov photons production and propagation





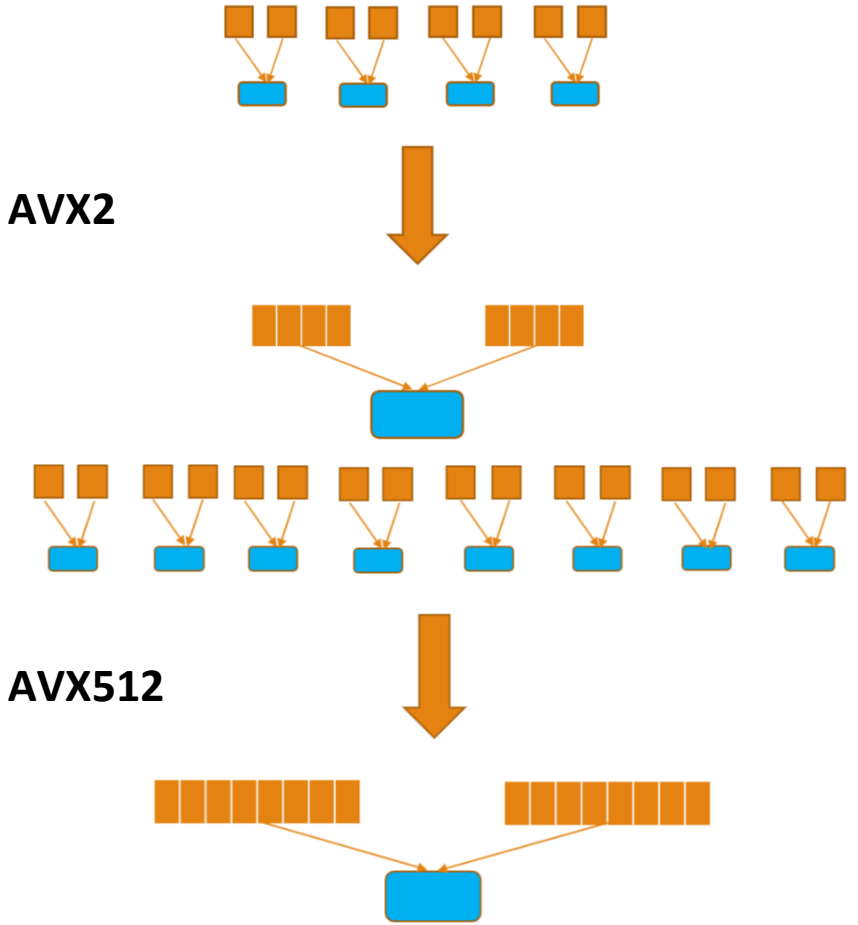
Profiling CORSIKA v7

- **Profiling on a 100 shower run.**
- **LONGI option enabled**
- **76.90% runtime in CERENK**
- **32.49% runtime in Raybnd**

Vectorization

- Having to compute a number of scalar operations on same type variables.

- Storing a number of variables in a vector and using a vectorized operation equivalent to the scalar one.

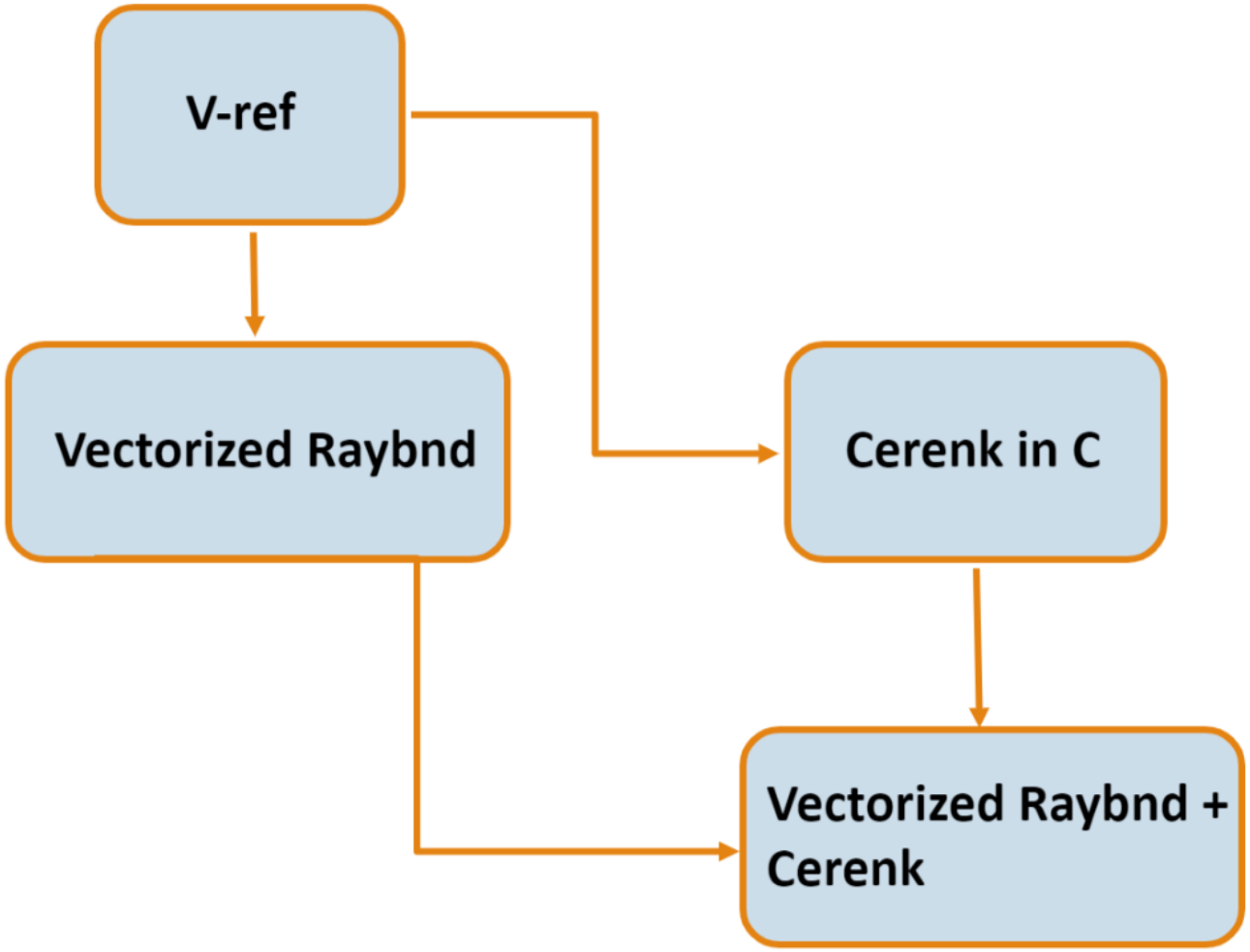


Vectorization

- Intel Intrinsics to manually vectorize the code .

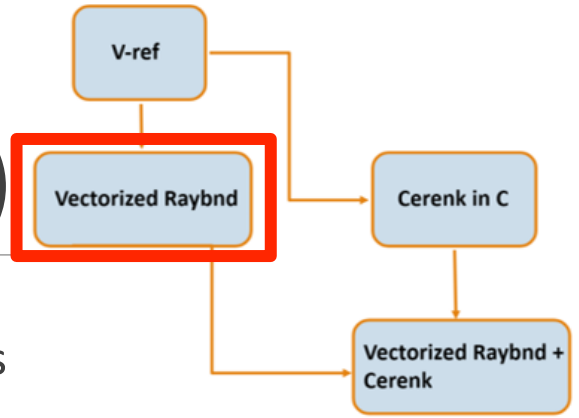
- External libraries that implement vectorized functions such as math libraries.
  - SIMD vector libm
    - https://gitlab.com/cquirin/vector-libm
    - https://hal.archives-ouvertes.fr/hal-01511131/document

- Automatic vectorization of the code by the compiler.



Optimization process

Optimization process (Raybnd)

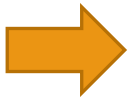


- Previous work on the vectorization Raybnd by computing mutliple photon bunches once.
  - https://gite.lirmm.fr/cta-optimization-group/cta-optimization-project/wikis/uploads/d9fe1652712cbfffe6a28c5d8c3a5da9/CorsikaOpt.pdf
- The vectorization of mathematical functions (exp, cos …) using a dedicated library
  - https://gitlab.com/cquirin/vector-libm
- Unrolling Cerenk Fortran loop to use vectorized Raybnd.
- Factorizing the code to avoid redundant function calls.

Optimization process (Raybnd)

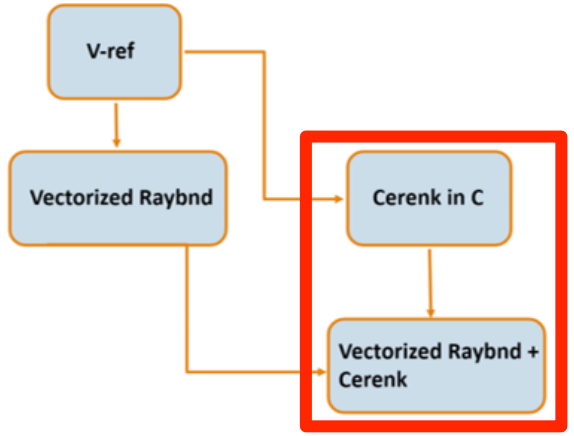
- Advancing the optimization of Raybnd by:
  - Restructuring the tests.
  - Isolating computations to facilitate detection of vectorizable loops by compiler.
- Validation of vectorization by checking assembler code.

```
*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;
```



```
for(int i=0; i< VECTOR_SIZE; i++){
  u[i] *= sin_t_obs[i]/sin_t_em[i];
}

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

for(int i=0; i< VECTOR_SIZE; i++){
  if(w[i] >= 0)
    w[i] = sqrt(1.-sin_t_obs[i]*sin_t_obs[i]);
  else
    w[i] = -sqrt(1.-sin_t_obs[i]*sin_t_obs[i]);
}

for(int i=0; i< VECTOR_SIZE; i++){
  dx[i] += hori_off[i] * (u[i])/sin_t_obs[i];
}

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

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

Optimization process (Cerenk)



- Optimization opportunity in the vectorization of unrolled loop in Cerenk.

- Translating the function from Fortran to C for easier application of the transformations applied to Raybnd.

- The Translation to C requires:
  - Defining the data structures mainly the Fortran common blocks, in C.
  - Making sure conditional expressions are equivalent between the two languages.
  - Linking the C version of Cerenk to the required functions still in Fortran.

Optimization process (Cerlde)

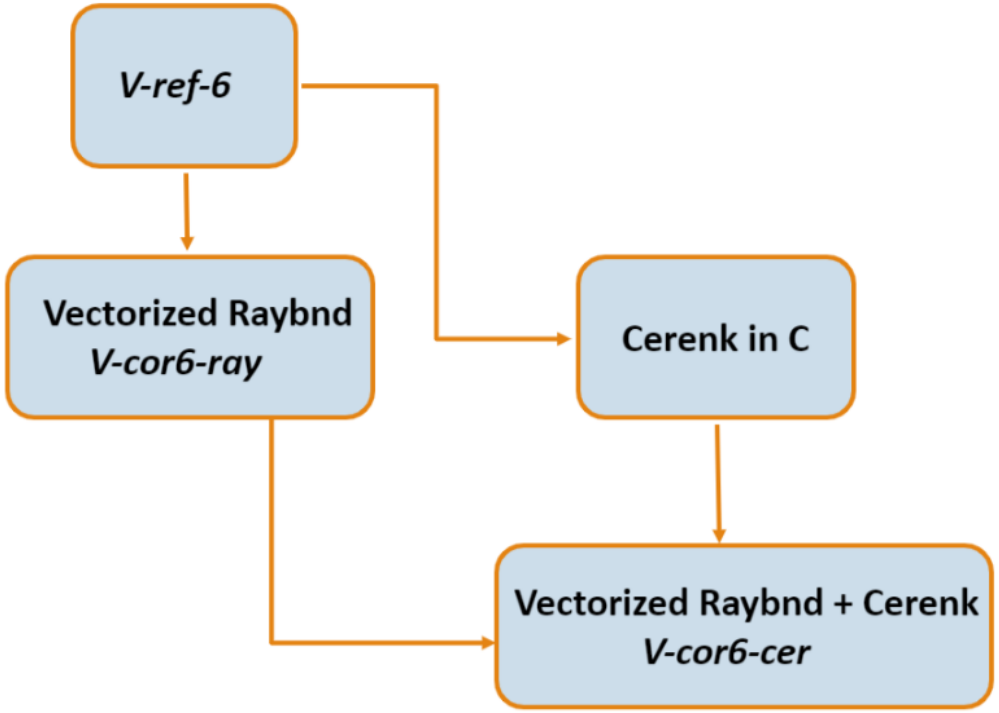
- LONGI option is impossible to activate with vectorized versions with scalar Cerlde.

⟹ A vectorized Cerlde function is necessary.

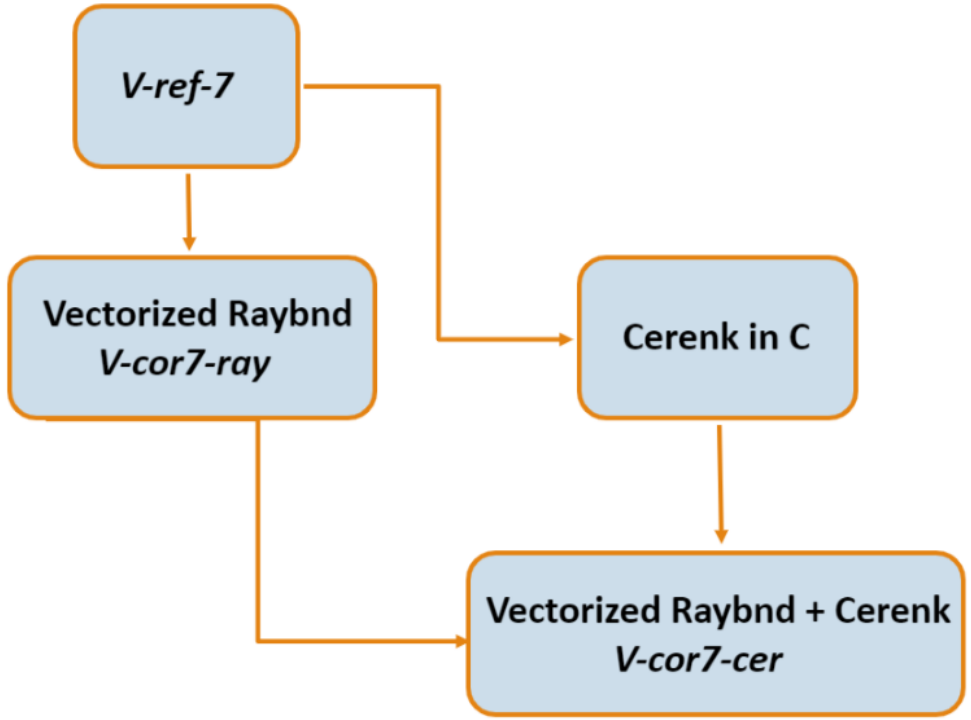
- Inlining Cerlde in the vectorized portion of CERENK while calling the sequential function in the computation of the remaining photon bunches.

Optimization process

Corsika v6 / IACT 1.51



Corsika v7 / IACT 1.59

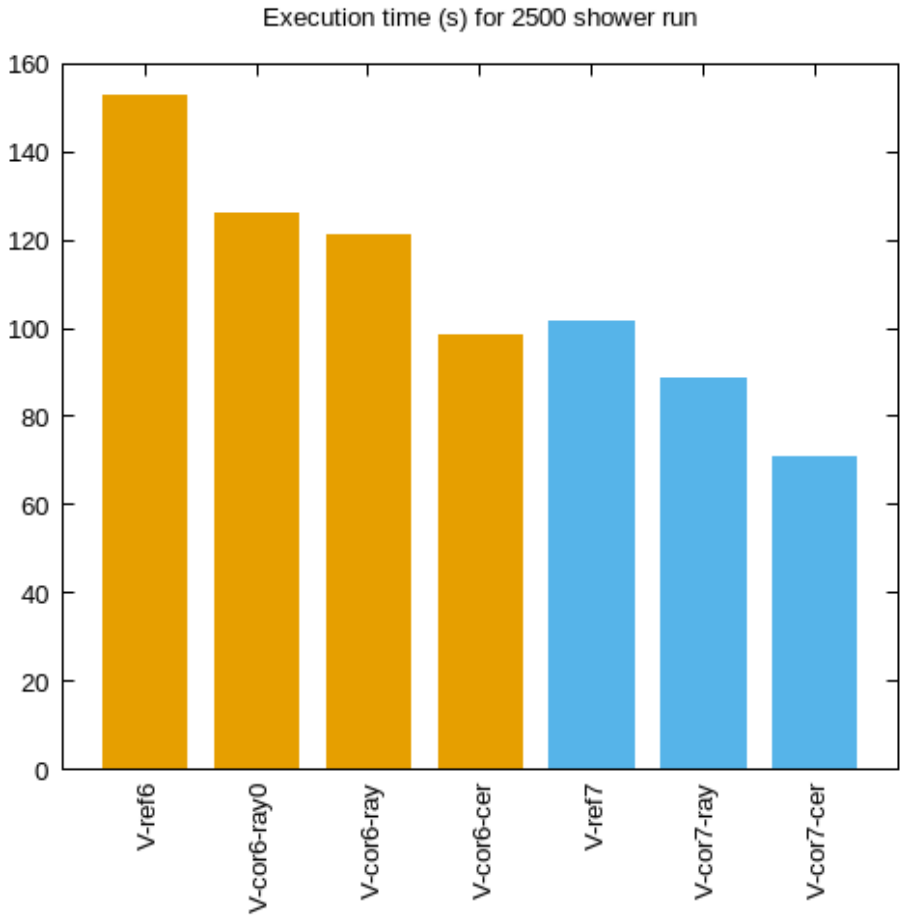


Experimental setup

- Dedicated server running running CentOS Linux release 7.4 - 64 bits.
- Compiler: **gcc 8.2.1**
- Compilation flags: **-O3 -mavx2**
- Running conditions:
  - gamma, prod3 Paranal baseline 20 deg. zenith angle (LONGI disabled)
  - Using keep-seeds option for random number generation to obtain reproductible runs
- Run validation using python script based on https://github.com/fact-project/pyeventio

```
Architecture:          x86_64
CPU op-mode(s):        32-bit, 64-bit
Byte Order:            Little Endian
CPU(s):                16
On-line CPU(s) list:   0-15
Thread(s) per core:    2
Core(s) per socket:    4
Socket(s):             2
NUMA node(s):          2
Vendor ID:             GenuineIntel
CPU family:            6
Model:                 85
Model name:            Intel(R) Xeon(R) Gold 5122 CPU @ 3.60GHz
Stepping:              4
CPU MHz:               3600.000
BogoMIPS:              7200.00
Virtualization:        VT-x
L1d cache:             32K
L1i cache:             32K
L2 cache:              1024K
L3 cache:              16896K
```

Experiments and Performance



	V-cor6-ray0	V-cor6-ray	V-cor6-cer
Speed-up	1.21	1.26	1.55

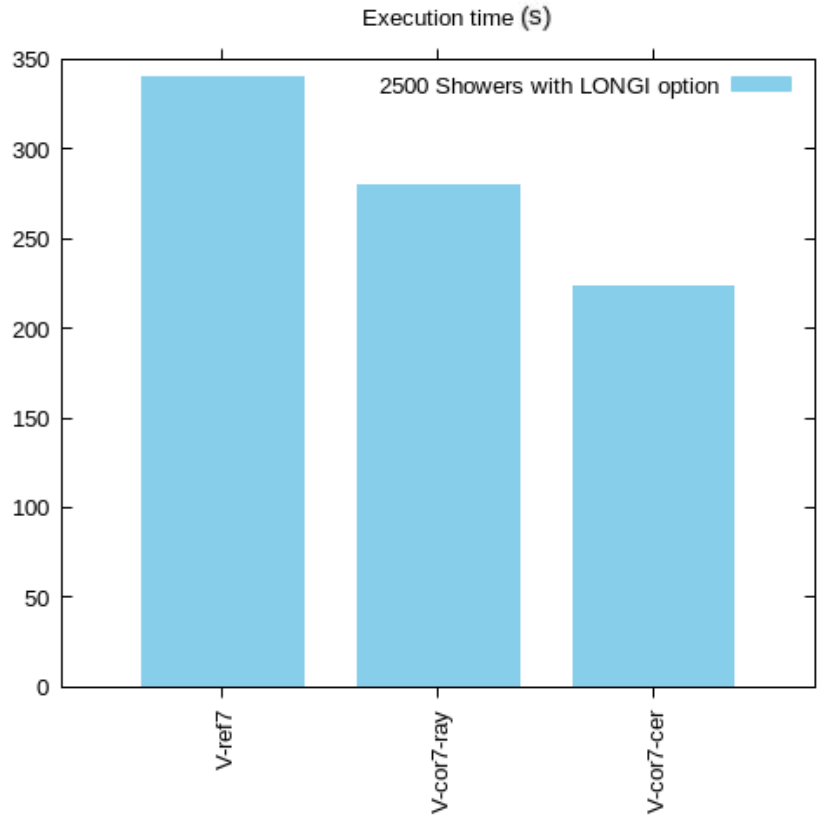
	V-cor7-ray	V-cor7-cer
Speed-up	1.14	1.44

- Less CPU time spent in Raybnd in Corsika v7 : New interpolation scheme.

Experiments and Performance

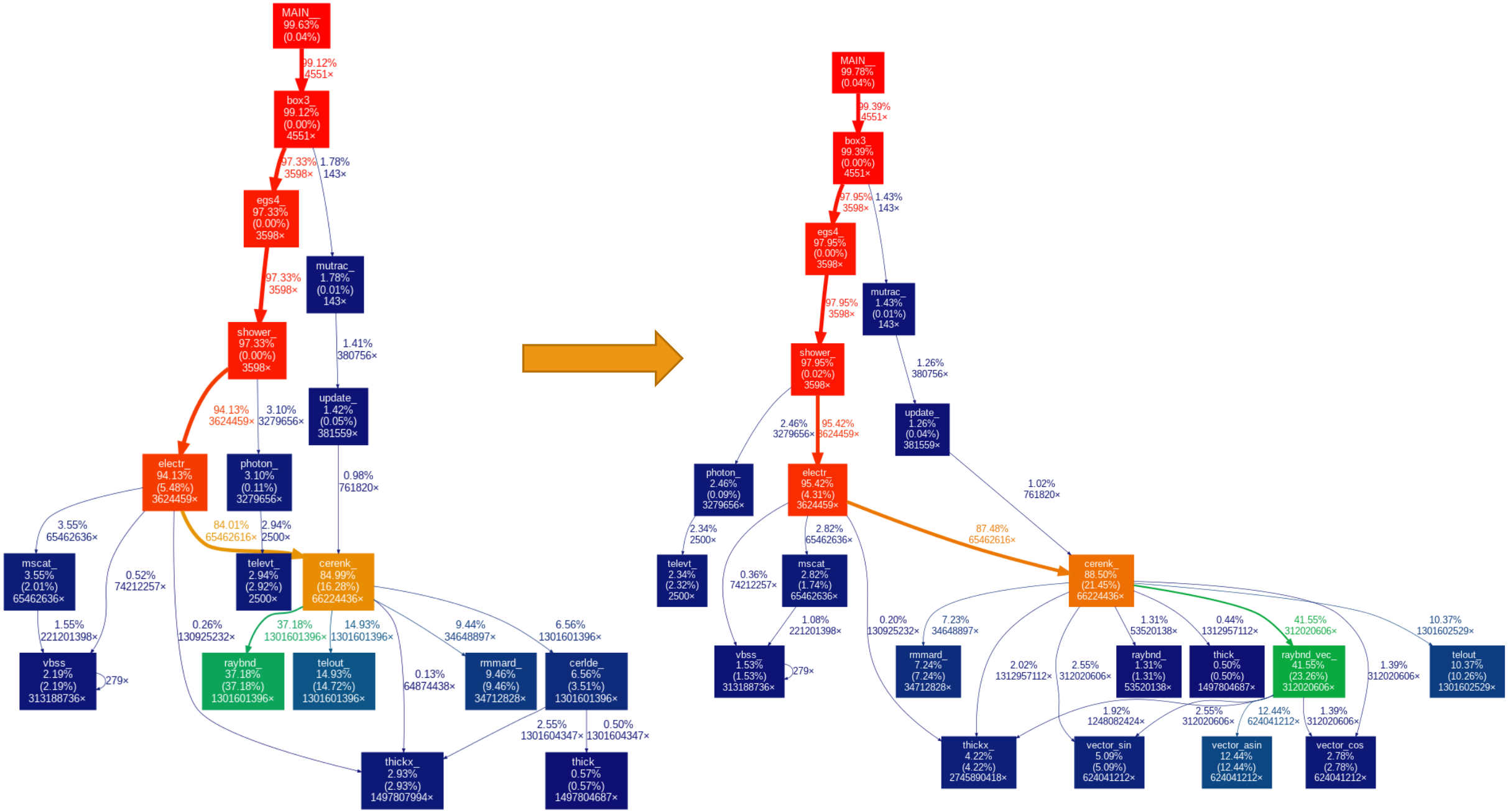
- The vectorization of CERLDE ⇒ A gain in performance.
  - Speedup of V-cor7-ray + LONGI option : **1.21**
  - Speedup of V-cor7-cer + LONGI option : **1.52**

- All previous experiments have vector length set to 4
  - Speedup of V-cor7-cer with –mavx512 flag (vector length = 8) : **1.60**



Conclusions

- The optimization of Corsika via vectorization:
  - Corsika v6 / IACT 1.51 a speedup of up to **1.55**
  - Corsika v7 / IACT 1.59 a speedup of up to **1.44** and **1.60** with vectors of 8 doubles.
- Corsika v7 / IACT 1.59 with LONGI option speedup : **1.52**
- Vectorized versions have identical results as the Ref versions so no accuracy is sacrificed.
- Optimization plans based on memory access and precision reduction are currently being investigated.
- The vectorized versions are in the process of being packaged for distribution.


MAIN_
99.63%
(0.04%)
99.12%
4551×
box3_
99.12%
(0.00%)
4551×
97.33%
3598×
1.78%
143×
egs4_
97.33%
(0.00%)
3598×
mutrac_
1.78%
(0.01%)
143×
97.33%
3598×
shower_
97.33%
(0.00%)
3598×
1.41%
380756×
update_
1.42%
(0.05%)
381559×
94.13%
3624459×
3.10%
3279656×
electr_
94.13%
(5.48%)
3624459×
photon_
3.10%
(0.11%)
3279656×
0.98%
761820×
3.55%
65462636×
84.01%
65462616×
2.94%
2500×
mscat_
3.55%
(2.01%)
65462636×
televt_
2.94%
(2.92%)
2500×
cerenk_
84.99%
(16.28%)
66224436×
0.52%
74212257×
1.55%
221201398×
vbss_
2.19%
(2.19%)
313188736×
279×
0.26%
130925232×
37.18%
1301601396×
14.93%
1301601396×
9.44%
34648897×
6.56%
1301601396×
raybnd_
37.18%
(37.18%)
1301601396×
telout_
14.93%
(14.72%)
1301601396×
0.13%
64874438×
rmmard_
9.46%
(9.46%)
34712828×
cerlde_
6.56%
(3.51%)
1301601396×
2.55%
1301604347×
0.50%
1301604347×
thickx_
2.93%
(2.93%)
1497807994×
thick_
0.57%
(0.57%)
1497804687×
MAIN_
99.78%
(0.04%)
99.39%
4551×
box3_
99.39%
(0.00%)
4551×
97.95%
3598×
1.43%
143×
egs4_
97.95%
(0.00%)
3598×
mutrac_
1.43%
(0.01%)
143×
97.95%
3598×
shower_
97.95%
(0.02%)
3598×
1.26%
380756×
update_
1.26%
(0.04%)
381559×
2.46%
3279656×
95.42%
3624459×
photon_
2.46%
(0.09%)
3279656×
electr_
95.42%
(4.31%)
3624459×
1.02%
761820×
2.34%
2500×
87.48%
65462616×
televt_
2.34%
(2.32%)
2500×
2.82%
65462636×
mscat_
2.82%
(1.74%)
65462636×
0.36%
74212257×
1.08%
221201398×
vbss_
1.53%
(1.53%)
313188736×
279×
0.20%
130925232×
cerenk_
88.50%
(21.45%)
66224436×
7.23%
34648897×
rmmard_
7.24%
(7.24%)
34712828×
1.31%
53520138×
raybnd_
1.31%
(1.31%)
53520138×
0.44%
1312957112×
thick_
0.50%
(0.50%)
1497804687×
41.55%
312020606×
raybnd_vec_
41.55%
(23.26%)
312020606×
10.37%
1301602529×
telout_
10.37%
(10.26%)
1301602529×
2.02%
1312957112×
2.55%
312020606×
1.39%
312020606×
1.92%
1248082424×
2.55%
312020606×
12.44%
624041212×
1.39%
312020606×
thickx_
4.22%
(4.22%)
2745890418×
vector_sin
5.09%
(5.09%)
624041212×
vector_asin
12.44%
(12.44%)
624041212×
vector_cos
2.78%
(2.78%)
624041212×