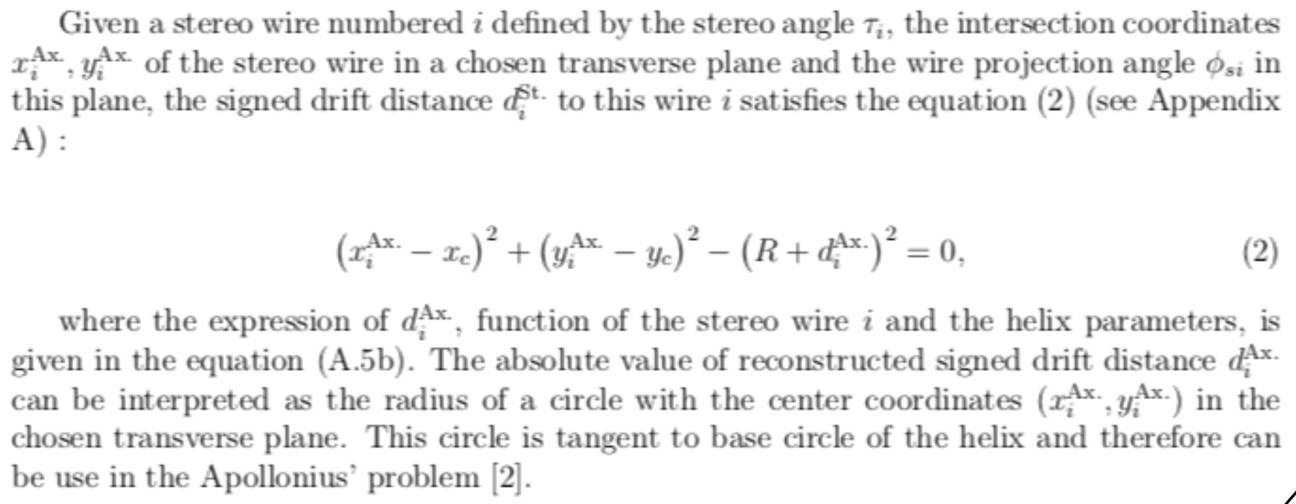
Patrice Lebrun

Tracking Meeting at Lyon - CC
October 30, 2024

Tracking_Apollonius in ICEDUST

A New Track/Hit Finding Approach

- Apollonius' problem extended to a **Full Stereo Drift Chamber**
- The method is described in the ArXiv https://arxiv.org/abs/2401.04576
- Uses **Julia** programming Language

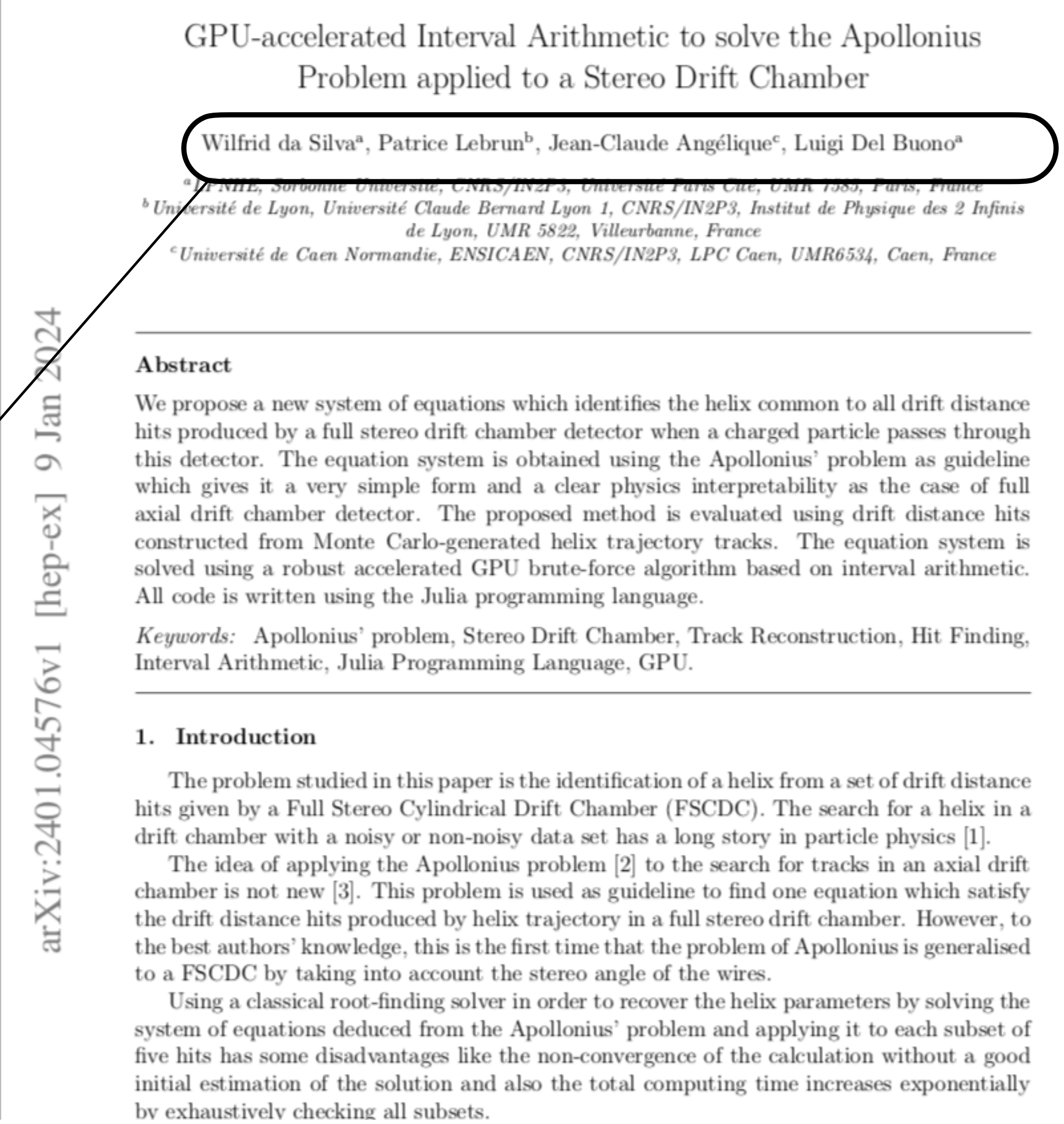


Given a stereo wire numbered i defined by the stereo angle τ_i, the intersection coordinates $x_i^{\mathrm{Ax.}}, y_i^{\mathrm{Ax.}}$ of the stereo wire in a chosen transverse plane and the wire projection angle ϕ_{si} in this plane, the signed drift distance $d_i^{\mathrm{St.}}$ to this wire i satisfies the equation (2) (see Appendix A) :

$$\left(x_i^{\mathrm{Ax.}} - x_c\right)^2 + \left(y_i^{\mathrm{Ax.}} - y_c\right)^2 - \left(R + d_i^{\mathrm{Ax.}}\right)^2 = 0, \qquad (2)$$

where the expression of $d_i^{\mathrm{Ax.}}$, function of the stereo wire i and the helix parameters, is given in the equation (A.5b). The absolute value of reconstructed signed drift distance $d_i^{\mathrm{Ax.}}$ can be interpreted as the radius of a circle with the center coordinates $(x_i^{\mathrm{Ax.}}, y_i^{\mathrm{Ax.}})$ in the chosen transverse plane. This circle is tangent to base circle of the helix and therefore can be use in the Apollonius' problem [2].

Wilfrid da Silva, Patrice Lebrun,
Jean-Claude Angélique, Luigi Del Buono



arXiv:2401.04576v1 [hep-ex] 9 Jan 2024

GPU-accelerated Interval Arithmetic to solve the Apollonius Problem applied to a Stereo Drift Chamber

Wilfrid da Silva[a], Patrice Lebrun[b], Jean-Claude Angélique[c], Luigi Del Buono[a]

[a] LPNHE, Sorbonne Université, CNRS/IN2P3, Université Paris Cité, UMR 7585, Paris, France
[b] Université de Lyon, Université Claude Bernard Lyon 1, CNRS/IN2P3, Institut de Physique des 2 Infinis de Lyon, UMR 5822, Villeurbanne, France
[c] Université de Caen Normandie, ENSICAEN, CNRS/IN2P3, LPC Caen, UMR6534, Caen, France

Abstract

We propose a new system of equations which identifies the helix common to all drift distance hits produced by a full stereo drift chamber detector when a charged particle passes through this detector. The equation system is obtained using the Apollonius' problem as guideline which gives it a very simple form and a clear physics interpretability as the case of full axial drift chamber detector. The proposed method is evaluated using drift distance hits constructed from Monte Carlo-generated helix trajectory tracks. The equation system is solved using a robust accelerated GPU brute-force algorithm based on interval arithmetic. All code is written using the Julia programming language.

Keywords: Apollonius' problem, Stereo Drift Chamber, Track Reconstruction, Hit Finding, Interval Arithmetic, Julia Programming Language, GPU.

1. Introduction

The problem studied in this paper is the identification of a helix from a set of drift distance hits given by a Full Stereo Cylindrical Drift Chamber (FSCDC). The search for a helix in a drift chamber with a noisy or non-noisy data set has a long story in particle physics [1].

The idea of applying the Apollonius problem [2] to the search for tracks in an axial drift chamber is not new [3]. This problem is used as guideline to find one equation which satisfy the drift distance hits produced by helix trajectory in a full stereo drift chamber. However, to the best authors' knowledge, this is the first time that the problem of Apollonius is generalised to a FSCDC by taking into account the stereo angle of the wires.

Using a classical root-finding solver in order to recover the helix parameters by solving the system of equations deduced from the Apollonius' problem and applying it to each subset of five hits has some disadvantages like the non-convergence of the calculation without a good initial estimation of the solution and also the total computing time increases exponentially by exhaustively checking all subsets.

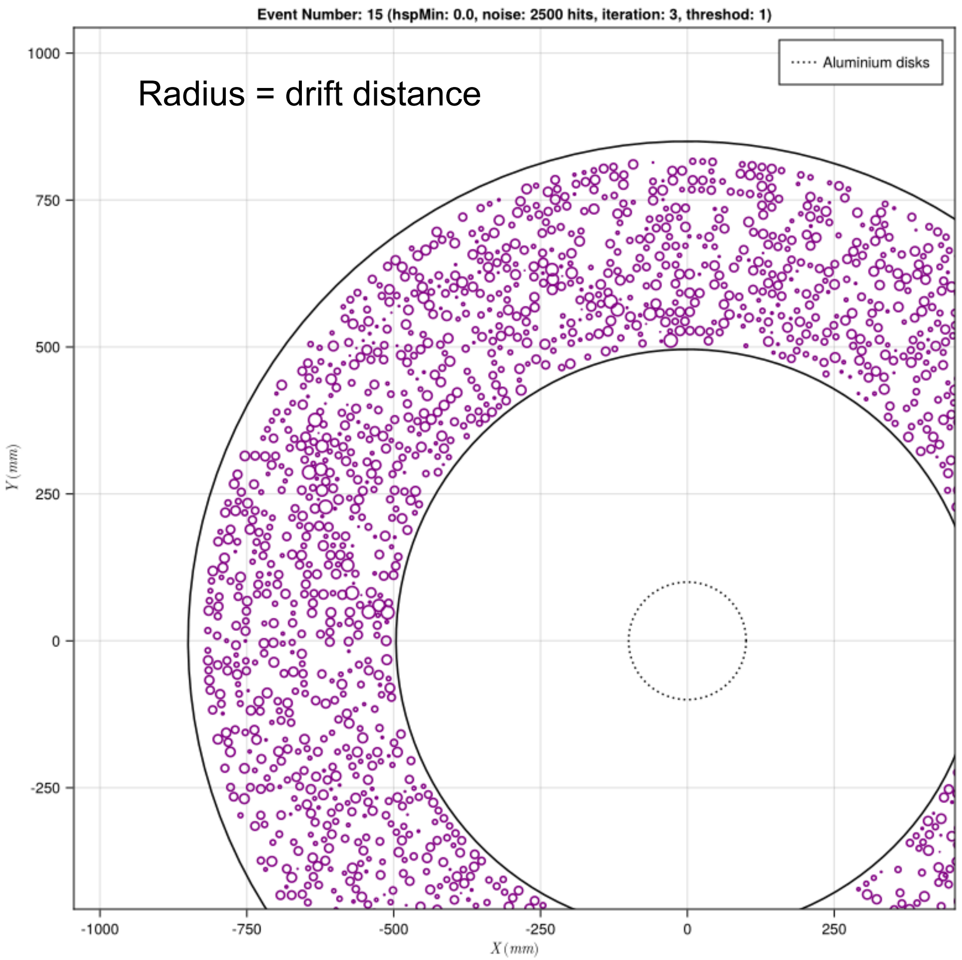
Goal and why Julia

- Find signal hits and tracks in a noisy environment with the **lowest possible processing time** and using **common computer hardware** like GPUs
- Improve efficiency of other methods.

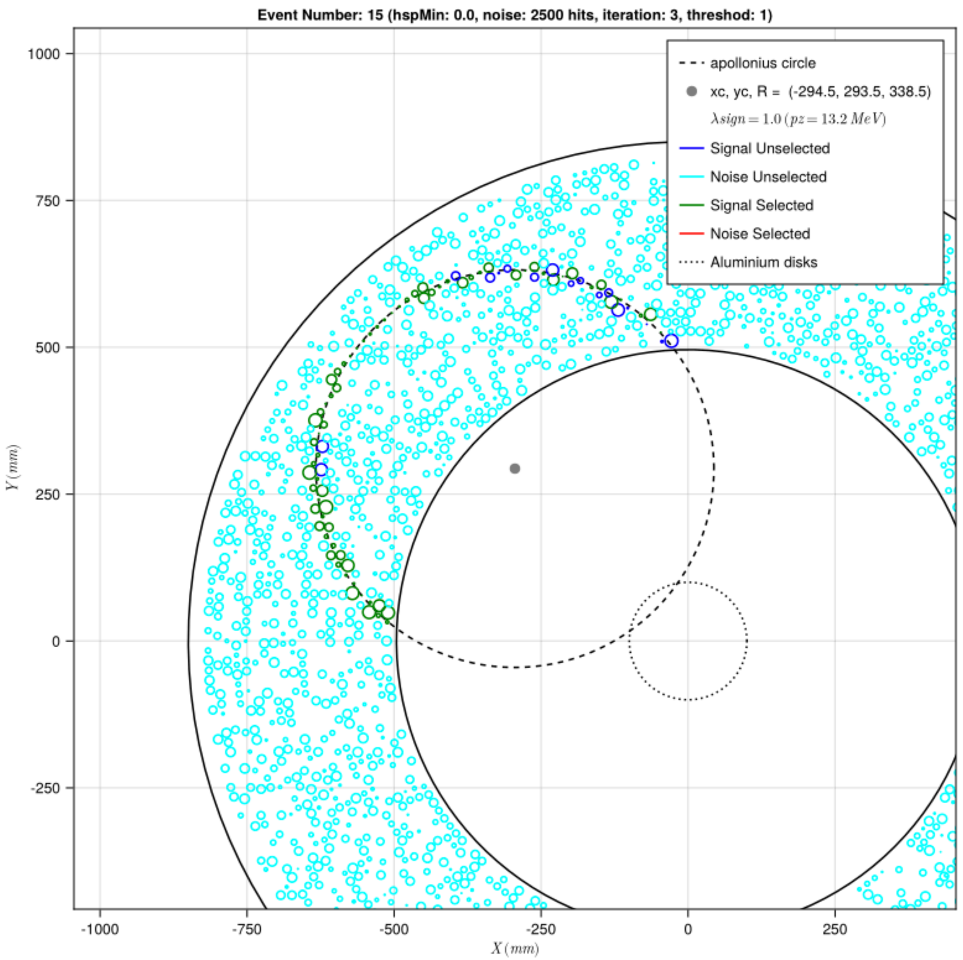
- Julia is a high-level, high-performance dynamic language for scientific and technical computing.
  - Allows an easy use of GPU with available powerful mathematical packages
  - Julia performance is identical to C/C++
  - Main packages for achieving this goal:
    - **CUDA.jl** (Nvidia)
    - **IntervalArithmetic.jl**

  - Useful Functionalities:
    - Managing Packages (Pkg) (installation, versioning, compatibilities, ...)
    - Embedding into C/C++
    - Broadcast and vectorization
    - Functions with Full dispatching
    - Garbage Collector (GC)
    - ...

illustration: event with noisy hits



Radius = drift distance



Plot w/ CairoMakie.jl package

Processing Time on NVIDIA V100: 2.55 ms/hit

Yao’s event

with *2500 hits of noise randomly distributed* ← Added using Toy_CDC.jl (our own Julia package to simulate the CDC)

- *The signal has* 89 hits
- 72 hits found (81%)
- 1 hit of noise selected

Development and Testing with Jupyter Notebooks

- Example with a starting part of a Jupyter notebook
  here is shown the nformation on packages and type of GPU used.
- More about Julia:
  - Compilations Just In Time (JIT) based on **LLVM**.
  - Very easy to install (and uninstall)
  - Open source with a large community (but << Python)
  - JuliaHEP community
  - Good documentation https://docs.julialang.org/en/v1/

```
[2]: import Pkg

[3]: Pkg.activate(".")

       Activating project at `~/comet/WilliamProject`

     Pkg.add(path="/pbs/home/l/lebrun/comet/dev_julia/Toy_CDC")

     Pkg.add(path="/pbs/home/l/lebrun/comet/dev_julia/NewApollonius")

[4]: Pkg.status()

     Status `~/comet/WilliamProject/Project.toml`
       [b507e648] NewApollonius v1.0.0-DEV `/pbs/home/l/lebrun/comet/dev_julia/NewApollonius#main`
       [7617a686] Toy_CDC v1.0.0 `/pbs/home/l/lebrun/comet/dev_julia/Toy_CDC#newGeometry`

[5]: Pkg.status(;mode=Pkg.PKGMODE_MANIFEST, outdated=true)

     Status `~/comet/WilliamProject/Manifest.toml`
     ⌅ [79e6a3ab] Adapt v3.7.2 (<v4.0.4): ArrayInterface, CUDA, GPUArrays, GPUArraysCore
     ⌅ [4fba245c] ArrayInterface v7.5.1 (<v7.10.0): julia
     ⌅ [ab4f0b2a] BFloat16s v0.2.0 (<v0.5.0): CUDA
     ⌅ [fa961155] CEnum v0.4.2 (<v0.5.0): CUDA, LLVM
     ^ [052768ef] CUDA v3.13.1 (<v5.3.4)
     ⌅ [6a86dc24] FiniteDiff v2.22.0 (<v2.23.1): julia
     ⌅ [0c68f7d7] GPUArrays v8.8.1 (<v10.1.1): CUDA
     ⌅ [46192b85] GPUArraysCore v0.1.5 (<v0.1.6): GPUArrays
     ⌅ [61eb1bfa] GPUCompiler v0.17.3 (<v0.26.4): CUDA
     ⌅ [d1acc4aa] IntervalArithmetic v0.20.8 (<v0.22.12): julia
     ^ [033835bb] JLD2 v0.4.47 (<v0.4.48)
     ⌅ [929cbde3] LLVM v4.17.1 (<v7.1.0): CUDA, GPUArrays, GPUCompiler
     ^ [b8f27783] MathOptInterface v1.29.0 (<v1.30.0)
     ^ [43287f4e] PtrArrays v1.1.0 (<v1.2.0)
     ^ [91c51154] SentinelArrays v1.4.2 (<v1.4.3)
     ^ [90137ffa] StaticArrays v1.9.3 (<v1.9.4)
     ^ [a759f4b9] TimerOutputs v0.5.23 (<v0.5.24)
     ⌅ [dad2f222] LLVMExtra_jll v0.0.18+0 (<v0.0.29+0): LLVM
     ^ [f50d1b31] Rmath_jll v0.4.0+0 (<v0.4.2+0)

[5]: import CUDA

[6]: CUDA.versioninfo()
     CUDA.devices()

     CUDA toolkit 11.7, artifact installation
     NVIDIA driver 465.19.1, for CUDA 11.3
     CUDA driver 11.3

     Libraries:
     - CUBLAS: 11.10.1
     - CURAND: 10.2.10
     - CUFFT: 10.7.2
     - CUSOLVER: 11.3.5
     - CUSPARSE: 11.7.3
     - CUPTI: 17.0.0
     - NVML: 11.0.0+465.19.1
     - CUDNN: 8.30.2 (for CUDA 11.5.0)
     - CUTENSOR: 1.4.0 (for CUDA 11.5.0)

     Toolchain:
     - Julia: 1.8.5
     - LLVM: 13.0.1
     - PTX ISA support: 3.2, 4.0, 4.1, 4.2, 4.3, 5.0, 6.0, 6.1, 6.3, 6.4, 6.5, 7.0, 7.1, 7.2
     - Device capability support: sm_35, sm_37, sm_50, sm_52, sm_53, sm_60, sm_61, sm_62, sm_70, sm_72, sm_75, sm_80, sm_86

     1 device:
       0: NVIDIA Tesla K80 (sm_37, 11.170 GiB / 11.173 GiB available)

[6]: CUDA.DeviceIterator() for 1 devices:
     0. NVIDIA Tesla K80
```

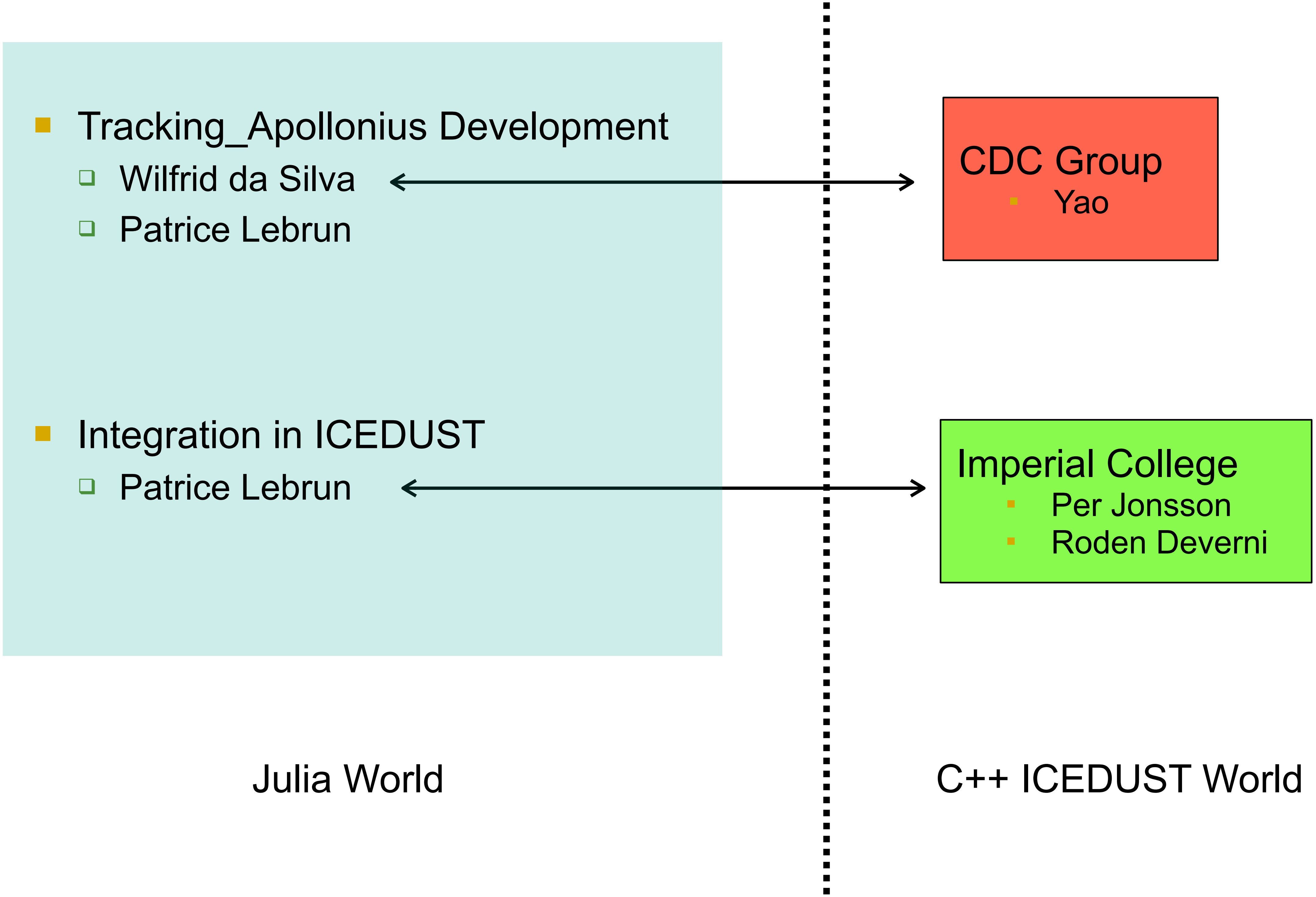
[052768ef] CUDA v3.13.1 (<v5.3.4)

Pinned Version to use old GPU K80, the only available on jupyterlab
New more powerful GPUs will be available in sept. (last version of CUDA will be used).

0. NVIDIA Tesla K80

Wikipedia: LLVM is a set of compiler and toolchain technologies[4] that can be used to develop a frontend for any programming language and a backend for any instruction set architecture.

Who



- Tracking_Apollonius Development
  - Wilfrid da Silva
  - Patrice Lebrun
- Integration in ICEDUST
  - Patrice Lebrun

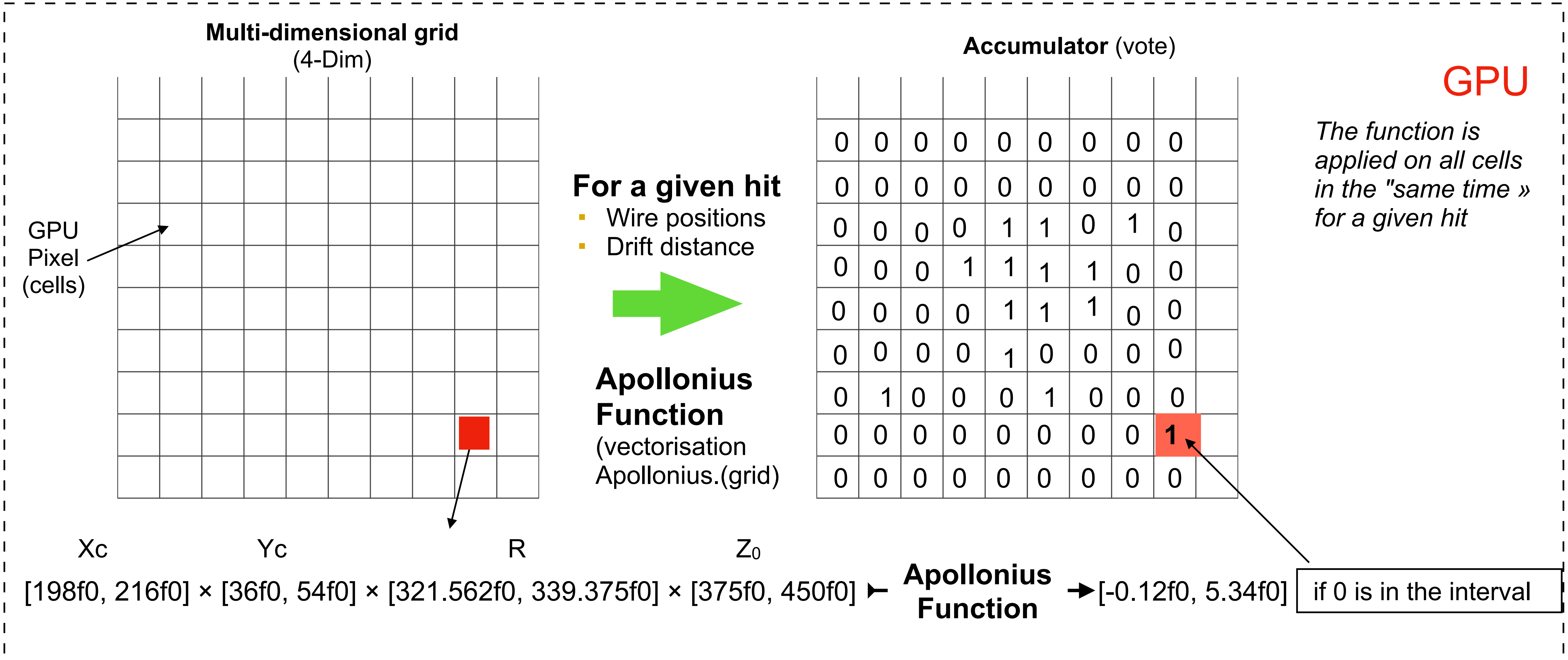
CDC Group
- Yao

Imperial College
- Per Jonsson
- Roden Deverni

Julia World

C++ ICEDUST World

Algorithm Based on Interval Arithmetic with GPU



Multi-dimensional grid
(4-Dim)

GPU
Pixel
(cells)

For a given hit
- Wire positions
- Drift distance

Apollonius Function
(vectorisation
Apollonius.(grid)

Accumulator (vote)

0	0	0	0	0	0	0	0	0	
0	0	0	0	0	0	0	0	0	
0	0	0	0	1	1	0	1	0	
0	0	0	1	1	1	1	0	0	
0	0	0	0	1	1	1	0	0	
0	0	0	0	1	0	0	0	0	
0	1	0	0	0	1	0	0	0	
0	0	0	0	0	0	0	0	**1**	
0	0	0	0	0	0	0	0	0	

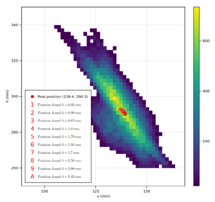
GPU

The function is applied on all cells in the "same time » for a given hit

Xc Yc R Z_0

[198f0, 216f0] × [36f0, 54f0] × [321.562f0, 339.375f0] × [375f0, 450f0] ⊢ **Apollonius Function** →[-0.12f0, 5.34f0]

if 0 is in the interval



Loop on all hits:
Sum of votes

Accumulator

···		22	18	···
···	32	31	32	···
	22	32	32	···
···	19	22	···	
	···	15	···	

Barycentre
(on a selection of cells)

Hits finding

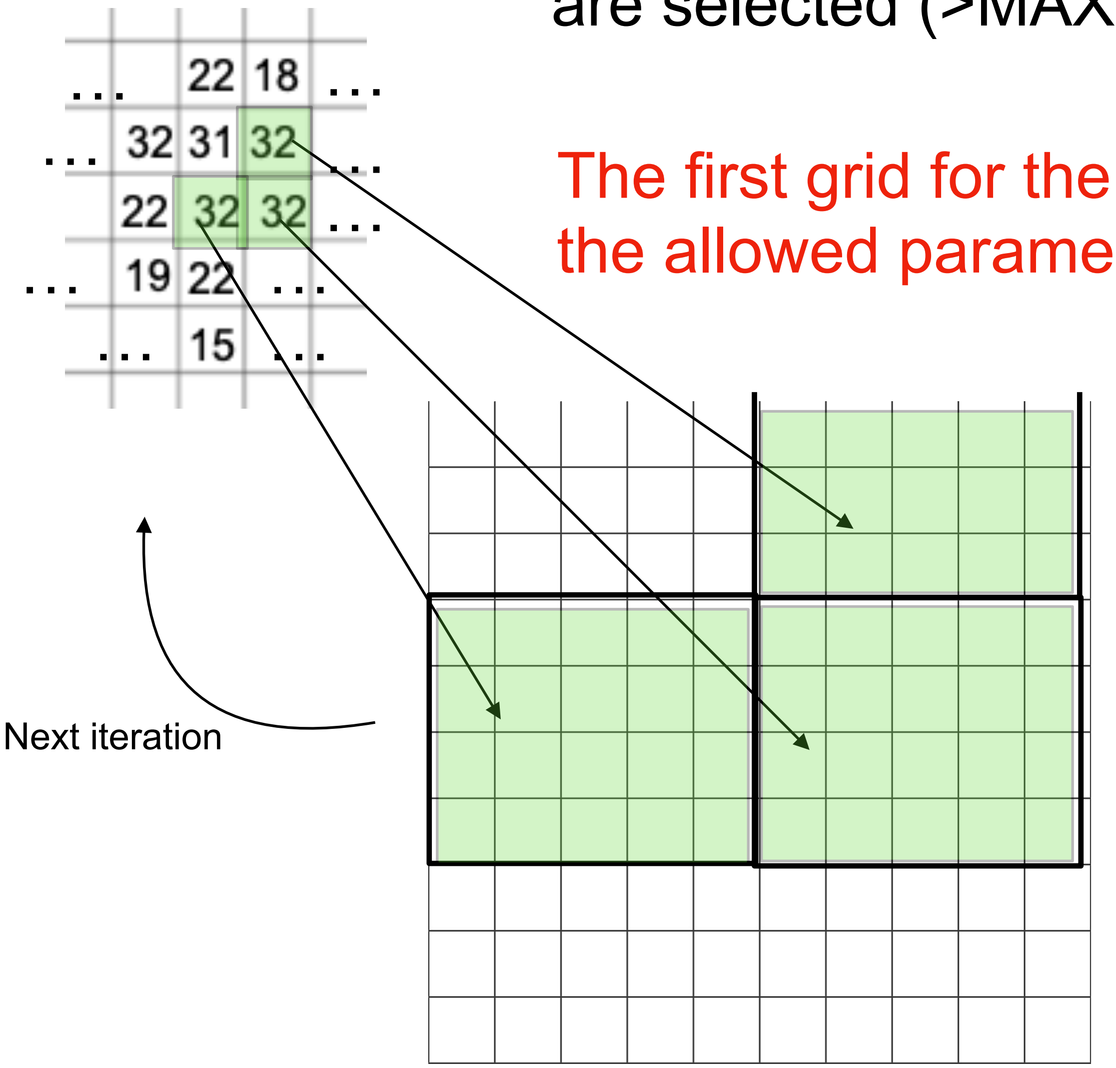
CPU

- Xc
- Yc
- R
- Z_0
- …
- **Probability Indicator on each hit**

Iterative Method

The cells having a number of votes greater than a given value are selected (>MAX-**threshold**) and subdivided to create a new grid.

The first grid for the first iteration has to cover all the allowed parameter space.



Example: 3 iterations (units: mm, MeV)

E=104.97
dE = 1.0e-5
intervals=[interval(-504.0, 504.0), interval(-504.0, 504.0), interval(87.0, 375.0), interval(0.0, 1500.0), interval(E-dE, E+dE)],
gridSubdivisions=[
    (ncellsX = 63, ncellsY = 63, ncellsR = 18, ncellsZ = 30, ncellsE = 1),
    (ncellsX=4, ncellsY=4, ncellsR=4, ncellsZ=4, ncellsE=1),
    (ncellsX=4, ncellsY=4, ncellsR=4, **ncellsZ=3**, ncellsE=1)]

With this configuration, the first grid has 2143260 cells.
Constraining the trajectory to be in the stopping target, the number of cells is reduced to 661620.
Number of cells in next grids depends on the event.
In the last grid, cell size : 1 x 1 x 1 x 4.2 mm^4 .

Equivalent to a single grid with ~ 32 billion of cells

- Hit finding consists to find hits having a solution in the selected cells (cells with value > threshold).
- **Only hits selected at the previous iteration are used for the next iteration.**
- Time processing is roughly proportional to the number of hits *(more noisy hits less time processing/hit)*

Parameters

- Can only be modified by a Tracking_Apollonius package expert
  - It could be necessary to decide whether some of them can be modified by ICEDUST users.

To use GPU in single or double precision:

```
"precision" => Float32,
```

Value of the uniform Magnetic field used by Apollonius

```
"magneticField" => 1.0f0,
```

If no cells has a number of votes greater or equal to this limit, the accumulator is empty (no track found).

```
"vote_min" => 15,
```

At each iteration, a set of thresholds is defined.
The highest value is used to select cells to create the grid for the next iteration.
A set of hits found correspond to a threshold (each threshold is associated to a probability to be a hit of signal).
A hit is found when it belongs to a cell with a number of votes > MAX(votes)-threshold

```
"thresholds_iter" => [[9], [1, 3, 5, 7], [1, 2, 3, 4, 5]],
```

Apollonius circle values are obtained with the cells having a number of votes > MAX(votes)-threshold_results.
For instance here, only the cells with the maximum of votes are used.

```
"threshold_results" => 1
```

Possible values domain of the Apollonius helix (Xc, Yc, R, Z0 and E)

```
"intervals" => IntervalArithmetic.Interval{Float32}
[[-504f0, 504f0], [-504f0, 504f0], [87f0, 375f0], [0f0, 1500f0], [104.969f0, 104.971f0]],
```

Defines the cells size for each iteration (number of cells in each dimension).

```
"subdivisions" => @NamedTuple{ncellsX::Int64, ncellsY::Int64, ncellsR::Int64, ncellsZ::Int64, ncellsE::Int64}[
(ncellsX = 63, ncellsY = 63, ncellsR = 18, ncellsZ = 30, ncellsE = 1),
(ncellsX = 4, ncellsY = 4, ncellsR = 4, ncellsZ = 4, ncellsE = 1),
(ncellsX = 4, ncellsY = 4, ncellsR = 4, ncellsZ = 3, ncellsE = 1)]
```

better to choose 1 (12.5 mm) or 2 (6.25 mm) ?
Is 4.1$\overline{6}$ mm not too small ?

Position of the projection plane

```
"zProj" => -791.626,
```

Performance

- Yao’s events Run001 (no noise added)
  - Number of events: 6754
  - Overall Total Number of Hits: 470710
  - 3.2 ms/hit on 1 device of NVIDIA V100 in single precision (this time per hit decreases when the number of noisy hits increases).
  - Resolutions [mm] (***assuming a 1 Tesla uniform magnetic field***).

	Mean	RMS
xc−xc0	−0.95	**7.71**
yc−yc0	−1.55	**8.53**
R−R0	0.67	**10.09**

  - Resolutions [mm] for differences lower than 20 mm in absolute value

	Mean	RMS	#events
xc−xc0	−0.74	**4,89**	6629 (98.1 %)
yc−yc0	−1.09	**5.73**	6552 (97.0 %)
R−R0	0.13	**5.33**	6478 (95.9 %)

  - Efficiencies:
    - Number of hits **not found** *(hit not in the accumulator)***:**
      - 22401 (4.75 %)
    - Number of hits **found** with the Highest Probability to be *a* hit of signal *(hit in cells with maximum of votes):*
      - 420068 (89.24 %)

Tracking_Apollonius in ICEDUST (1)

- ICEDUST_externals_source_LFS
  - Official Julia binary Release
  - it’s not recommended to use an own build of Julia (CUDA.jl)
    - Currently JULIA/julia-1.10.5-linux-x86_64.tar.gz (64 bits)
    - Other releases are available for many architectures and OS: https://julialang.org/downloads/
- ICEDUST_externals_install
  - julia-1.10.5-linux-x86_64 [**bin etc include lib libexec** LICENSE.md **share]**

- ICEDUST_packages
  - oaJuliaInterface (can be used for other Julia project)
    - Goal: ICEDUST user does not need to know Julia
    - C++ class with members to call some Tracking_Apollonius functions (currently)
    - Tracking_Apollonius is a submodule
    - Tracking_Apollonius **has to be a git repository** to be used by the package manager of Julia
      - Meaning Tracking_Apollonius can be used also in a pure Julia environment (important for development)

- ICEDUST_install
  - **julia_depot** directory where all packages and artefacts are installed (~2.7 Go)
  - Shared library *lib/liboaJuliaInterface.so* and setups and executables for testing are in oaJuliaInterface/bin
  - Setup.sh is updated

Tracking_Apollonius in ICEDUST (2)

- Use embedding Julia in C/C++
  - Simple functions are defined in Tracking_Apollonius to be easily called by ICEDUST, avoiding to have very sophisticated C/C++ code to write.
    - Limited to Array and Structure with leaf types (Int, float , double, bool …)

- Build procedures (new when red)
  - ICEDUST_externals_install

```
git clone  git@gitlab.in2p3.fr:Patrice/ICECDUST_external_sources_LFS.git
cd ..; mkdir ext_build; cd ext_build
cmake -DBUILD_JULIA=TRUE -DBUILD_GEANT4_VERSION=4.10.6  ../ICECDUST_external_sources_LFS
make -j4
```

‘Patrice’ is temporary, it should be comet (hoping so)

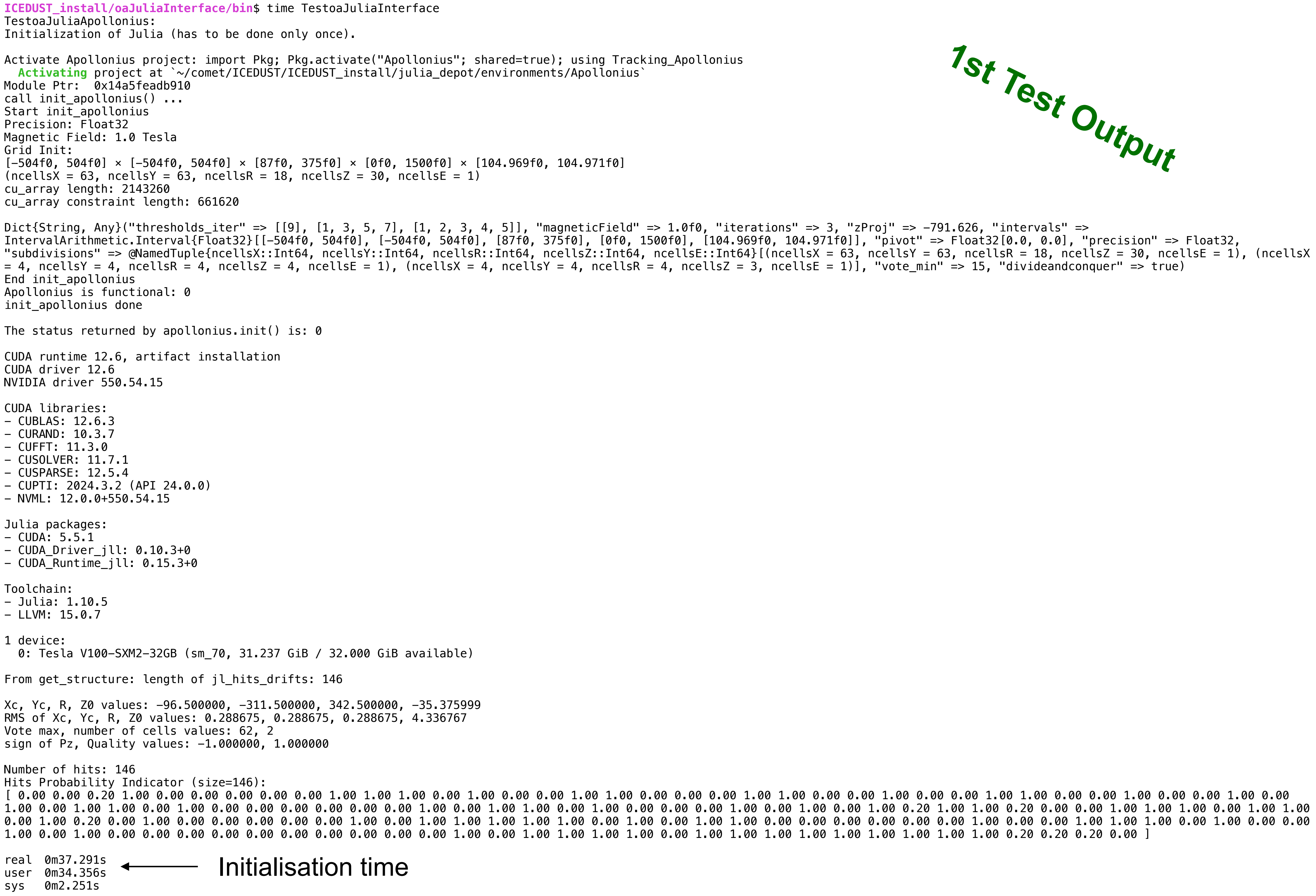
  - ICEDUST_install
    - build has to be done on machine with GPU (the use of docker is underway).

```
git clone  git@gitlab.in2p3.fr:Patrice/ICEDUST_packages.git
cd ICEDUST_packages
git clone git@gitlab.in2p3.fr:Patrice/Tracking_Apollonius
cd ..; mkdir build; cd build
cmake -DBUILD_JULIA=TRUE -DBUILD_GEANT4_VERSION=4.10.6  ../ICEDUST_packages
make -j4
```

Notes:
git clone --recurse-submodules git@gitlab.in2p3.fr:Patrice/ICEDUST_packages.git
will replace the two git clone commands when julia will be able to use submodules
(a patch is available but not yet applied).

Testing

- Two executables are available in oaJuliaInterface/bin for testing and measuring time performance
  - 1st Test (time TestoaJuliaInterface)
    - check the result of Tracking_Apollonius with a sample of hits
    - Measure the initialisation time
  - 2st Test (time TestoaJuliaInterfaceLoop <int>)
    - to measure processing time per hit
  - The sources are in ICEDUST_packages/oaJuliaInterface/app
    - Could be used as example to use the interface.



```
ICEDUST_install/oaJuliaInterface/bin$ time TestoaJuliaInterface
TestoaJuliaApollonius:
Initialization of Julia (has to be done only once).

Activate Apollonius project: import Pkg; Pkg.activate("Apollonius"; shared=true); using Tracking_Apollonius
  Activating project at `~/comet/ICEDUST/ICEDUST_install/julia_depot/environments/Apollonius`
Module Ptr:  0x14a5feadb910
call init_apollonius() ...
Start init_apollonius
Precision: Float32
Magnetic Field: 1.0 Tesla
Grid Init:
[-504f0, 504f0] × [-504f0, 504f0] × [87f0, 375f0] × [0f0, 1500f0] × [104.969f0, 104.971f0]
(ncellsX = 63, ncellsY = 63, ncellsR = 18, ncellsZ = 30, ncellsE = 1)
cu_array length: 2143260
cu_array constraint length: 661620

Dict{String, Any}("thresholds_iter" => [[9], [1, 3, 5, 7], [1, 2, 3, 4, 5]], "magneticField" => 1.0f0, "iterations" => 3, "zProj" => -791.626, "intervals" =>
IntervalArithmetic.Interval{Float32}[[-504f0, 504f0], [-504f0, 504f0], [87f0, 375f0], [0f0, 1500f0], [104.969f0, 104.971f0]], "pivot" => Float32[0.0, 0.0], "precision" => Float32,
"subdivisions" => @NamedTuple{ncellsX::Int64, ncellsY::Int64, ncellsR::Int64, ncellsZ::Int64, ncellsE::Int64}[(ncellsX = 63, ncellsY = 63, ncellsR = 18, ncellsZ = 30, ncellsE = 1), (ncellsX
= 4, ncellsY = 4, ncellsR = 4, ncellsZ = 4, ncellsE = 1), (ncellsX = 4, ncellsY = 4, ncellsR = 4, ncellsZ = 3, ncellsE = 1)], "vote_min" => 15, "divideandconquer" => true)
End init_apollonius
Apollonius is functional: 0
init_apollonius done

The status returned by apollonius.init() is: 0

CUDA runtime 12.6, artifact installation
CUDA driver 12.6
NVIDIA driver 550.54.15

CUDA libraries:
- CUBLAS: 12.6.3
- CURAND: 10.3.7
- CUFFT: 11.3.0
- CUSOLVER: 11.7.1
- CUSPARSE: 12.5.4
- CUPTI: 2024.3.2 (API 24.0.0)
- NVML: 12.0.0+550.54.15

Julia packages:
- CUDA: 5.5.1
- CUDA_Driver_jll: 0.10.3+0
- CUDA_Runtime_jll: 0.15.3+0

Toolchain:
- Julia: 1.10.5
- LLVM: 15.0.7

1 device:
  0: Tesla V100-SXM2-32GB (sm_70, 31.237 GiB / 32.000 GiB available)

From get_structure: length of jl_hits_drifts: 146

Xc, Yc, R, Z0 values: -96.500000, -311.500000, 342.500000, -35.375999
RMS of Xc, Yc, R, Z0 values: 0.288675, 0.288675, 0.288675, 4.336767
Vote max, number of cells values: 62, 2
sign of Pz, Quality values: -1.000000, 1.000000

Number of hits: 146
Hits Probability Indicator (size=146):
[ 0.00 0.00 0.20 1.00 0.00 0.00 0.00 0.00 0.00 1.00 1.00 1.00 0.00 1.00 0.00 0.00 1.00 1.00 0.00 0.00 0.00 1.00 1.00 0.00 0.00 1.00 0.00 0.00 1.00 1.00 0.00 0.00 1.00 0.00 0.00 1.00 0.00
1.00 0.00 1.00 1.00 0.00 1.00 0.00 0.00 0.00 0.00 0.00 0.00 1.00 0.00 1.00 1.00 0.00 1.00 0.00 0.00 0.00 1.00 0.00 1.00 0.00 1.00 0.20 1.00 1.00 0.20 0.00 0.00 1.00 1.00 1.00 0.00 1.00 1.00
0.00 1.00 0.20 0.00 1.00 0.00 0.00 0.00 0.00 0.00 1.00 0.00 1.00 1.00 1.00 1.00 1.00 0.00 1.00 0.00 0.00 1.00 1.00 0.00 0.00 0.00 0.00 0.00 0.00 1.00 0.00 0.00 1.00 1.00 1.00 0.00 1.00 0.00 0.00
1.00 0.00 1.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 1.00 0.00 1.00 1.00 1.00 1.00 0.00 1.00 1.00 1.00 1.00 1.00 1.00 1.00 1.00 1.00 0.20 0.20 0.20 0.00 ]

real	0m37.291s
user	0m34.356s
sys	0m2.251s
```

1st Test Output

Initialisation time

2nd Test Output (100 loops)

```
ICEDUST_install/oaJuliaInterface/bin$ time TestoaJuliaInterfaceLoop 100
TestoaJuliaInterfaceLoop:
Number of loop: 100 ...
Initialization of Julia (has to be done only once).

Activate Apollonius project: import Pkg; Pkg.activate("Apollonius"; shared=true); using Tracking_Apollonius
  Activating project at `~/comet/ICEDUST/ICEDUST_install/julia_depot/environments/Apollonius`
Module Ptr:  0x14b3f07d1910
call init_apollonius() ...
Start init_apollonius
Precision: Float32
Magnetic Field: 1.0 Tesla
Grid Init:
[-504f0, 504f0] × [-504f0, 504f0] × [87f0, 375f0] × [0f0, 1500f0] × [104.969f0, 104.971f0]
(ncellsX = 63, ncellsY = 63, ncellsR = 18, ncellsZ = 30, ncellsE = 1)
cu_array length: 2143260
cu_array constraint length: 661620

Dict{String, Any}("thresholds_iter" => [[9], [1, 3, 5, 7], [1, 2, 3, 4, 5]], "magneticField" => 1.0f0, "iterations" => 3, "zProj" =>
-791.626, "intervals" => IntervalArithmetic.Interval{Float32}[[-504f0, 504f0], [-504f0, 504f0], [87f0, 375f0], [0f0, 1500f0],
[104.969f0, 104.971f0]], "pivot" => Float32[0.0, 0.0], "precision" => Float32, "subdivisions" => @NamedTuple{ncellsX::Int64,
ncellsY::Int64, ncellsR::Int64, ncellsZ::Int64, ncellsE::Int64}[(ncellsX = 63, ncellsY = 63, ncellsR = 18, ncellsZ = 30, ncellsE = 1),
(ncellsX = 4, ncellsY = 4, ncellsR = 4, ncellsZ = 4, ncellsE = 1), (ncellsX = 4, ncellsY = 4, ncellsR = 4, ncellsZ = 3, ncellsE = 1)],
"vote_min" => 15, "divideandconquer" => true)
End init_apollonius
Apollonius is functional: 0
init_apollonius done

The status returned by apollonius.init() is: 0
From get_structure: length of jl_hits_drifts: 146
From get_structure: length of jl_hits_drifts: 146
...
...
From get_structure: length of jl_hits_drifts: 146
From get_structure: length of jl_hits_drifts: 146
From get_structure: length of jl_hits_drifts: 146

real	1m19.585s
user	1m17.655s
sys	0m2.109s
```

Time processing per hit: ~3.1 ms (on V100 GPU type)

CODE

oaJuliaInterface/src/ICDChitsICEDUST.hxx

```
#ifndef OAJULIAINTERFACE_CDChitsICEDUST_HXX
#define OAJULIAINTERFACE_CDChitsICEDUST_HXX

#include <vector>
typedef struct
{
        std::vector<float>* xstarts;
        std::vector<float>* ystarts;
        std::vector<float>* zstarts;
        std::vector<float>* xends;
        std::vector<float>* yends;
        std::vector<float>* zends;
        std::vector<float>* drifts;
}CDChitsICEDUST;
```

Hits Structure:

Wire positions and drift distances
in the local system of coordinates of the CDC (unit in mm)
Tracking_Apollonius does not know anything about the geometry

```
#include <julia.h>
JULIA_DEFINE_FAST_TLS // only define this once, in an executable (not in a shared library) if you want fast code.

#include <IApollonius.hxx>
#include <ICDChitsICEDUST.hxx>

int main()
{
    IApollonius* apollonius = IApollonius::getInstance();  //singleton because a julia module can be seen as workspace
    // apollonius->CUDA_versioninfo();  //to print information on CUDA and GPU

    int status = apollonius->init();
    if (status != 0) return status;  // 0 means everything is ok

    CDChitsICEDUST hits;
    fill_hits(&hits);  // A function which fill the hits values

    const std::vector<float>* results = apollonius->apolloniusresults(&hits);
    // work on results (next slide) ...
}
```

Another function to save time processing:

```
const std::vector<float>* results = apollonius->apolloniusresultswithselection(&hits, &selection);
where selection is a vector<bool>
Only hits with a true value are used at the first iteration. All hits are used for hits finding.
```

Results

- const std::vector<float>* results = apollonius->apolloniusresults(&hits);

```
results vector:

     0: xc
     1: yc
     2: R
     3: Z0
     4: RMS xc
     5: RMS yc
     6: RMS R
     7: RMs Z0
     8: Vote Max
     9: Number of cells in the accumulator
     10: Sign of Pz
     11: Quality (iteration level)
     12: nhits (Number of hits)
     13:12+nhits: Probability Indicator for each hit given in input (hit found for a given threshold).
     13+nhits:end Sign of each drift distance (-1.0, 0.0 , 1.0 , 99.0 for hit flagged as noise)

 Quality is related to the iteration number of the given results.
 Probability Indicator is a probability for a hit to belong to the track signal.
```

Results

- Some Function Members are available to get partial informations:
  - the result on the current event is stored in the apollonius object.

```
    printf("Xc, Yc, R, Z0 values: %f, %f, %f, %f\n", apollonius->Xc(), apollonius->Yc(), apollonius->R(), apollonius->Z0());
    printf("RMS of Xc, Yc, R, Z0 values: %f, %f, %f, %f\n", apollonius->RMS_Xc(), apollonius->RMS_Yc(), apollonius->RMS_R(), 
apollonius->RMS_Z0());

    printf("Vote max, number of cells values: %d, %d\n",apollonius->vote_max(), apollonius->numberOfCells());
    printf("sign of Pz, Quality values: %f, %f\n", apollonius->PzSign(), apollonius->quality());

    printf("Number of hits: %d\n", apollonius->numberOfHits());


    std::vector<float> hitsProbabilityIndicator = apollonius->hitsProbabilityIndicator();
    printf("Hits Probability Indicator (size=%ld):\n", hitsProbabilityIndicator.size());
    printf("[ ");
    for(size_t i=0; i<hitsProbabilityIndicator.size(); i++)
        printf("%.2f ", hitsProbabilityIndicator[i]);
    printf(“]\n");

    std::vector<float> hitsDriftSigns = apollonius->hitsDriftSigns();
    printf("Hits Drift Signs (size=%ld):\n", hitsDriftSigns.size());
    printf("[ ");
    for(size_t i=0; i<hitsDriftSigns.size(); i++)
        printf("%.2f ", hitsDriftSigns[i]);
    printf("]\n");
```

Possible Future

- Integrating ICEDUST or a part of it in Julia
  - Why not if Julia has a lot of success

- An example of Integration in high energy physics .
  - Geant4 is used like a package by Julia.
  - Among other conclusion: Geant4.jl can be a very useful add-on to the Geant4 project
    - Tutorials (very easy to setup and portable), interactive development (notebooks), connection to other powerful packages in the Julia ecosystem (visualization, analysis, etc.)

2023 JuliaHEP Workshop

Geant4.jl - New Interface to Simulation Applications

Pere Mato/CERN
9 November 2023

https://github.com/JuliaHEP/Geant4.jl

Click on it to get he PDF

- JuliaHEP is working on ROOT integration

- UnROOT.jl is a packages already available to work with TTree