

# On SpdRoot profiling

Didorenko Alexey  
didorenko@jinr.ru

Voytishin Nikolay  
nvoytish@jinr.ru

MLIT JINR

# Relevance

## Spin Physics Detector



International spin physics collaboration at the collider NICA

[General information](#) [Collaboration](#) [Presentations and publications](#) [Setup](#) [Internal access](#)

### General Information

[SPD CDR & TDR](#)

[NEWS AND ANNOUNCEMENTS](#)

[UPCOMING CONFERENCES](#)

[CONTACTS](#)

[USEFUL LINKS](#)

### Collaboration

[PARTICIPATING INSTITUTIONS](#)

[EXECUTIVE BOARD](#)

[TECHNICAL BOARD](#)

[PUBLICATION COMMITTEE](#)

[DOCUMENTS](#)

### SPD Presentations

## SPD Software

### [SPD Software Wiki](#)

Monte Carlo simulation, event reconstruction for both simulated and real data, data analysis and visualization are planned to be performed by an object oriented C++ toolkit SPDroot. It is based on the FairRoot framework initially developed for the FAIR experiments at GSI Darmstadt and partially compatible with MPDroot and BM@Nroot software used at MPD and BM@N, respectively.

The SPD detector description for Monte Carlo simulation is based on the ROOT geometry while transportation of secondary particles through material of the setup and simulation of detector response is provided by GEANT4 code. The standard multipurpose generators like Pythia6 and Pythia8 as well as specialised generators can be used for simulation of primary nucleon-nucleon collision.

- [GIT Repository](#).

SpdRoot is a software package that is capable of performing Monte Carlo simulation of events, reconstruction, analysis and visualization of events.

It is argued that the reconstruction runs slower than expected event processing speeds.

The current issue of this project is to find bottlenecks in the source code of the program and further improve the processing speed and efficiency of computing resources.

# Aim of the work

**Aim of the work:** to find bottlenecks of the event reconstruction process in the SpdRoot source code.

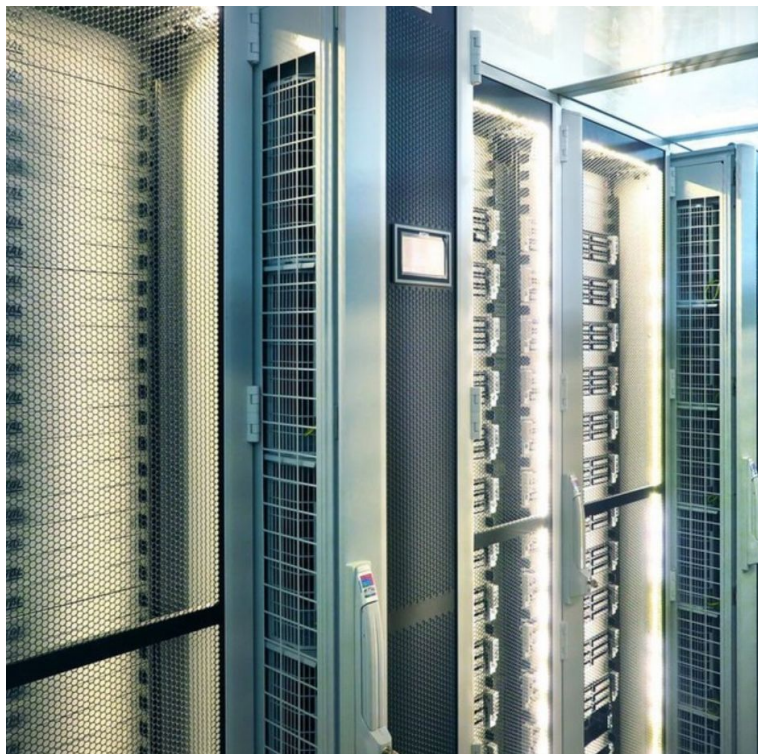
**Tasks:**

For reconstruction functions measure:

- % resources used;
- execution time.

To study the influence of field type on reconstruction speed.

# Technology stack



```
laxddid@ncx104 ~]$ perf

usage: perf [--version] [--help] [OPTIONS] COMMAND [ARGS]

The most commonly used perf commands are:
  annotate      Read perf.data (created by perf record) and display annotated code
  archive      Create archive with object files with build-ids found in perf.data file
  bench        General framework for benchmark suites
  buildid-cache Manage build-id cache.
  buildid-list List the buildids in a perf.data file
  c2c          Shared Data C2C/HITM Analyzer.
  config       Get and set variables in a configuration file.
  data         Data file related processing
  diff         Read perf.data files and display the differential profile
  evlist       List the event names in a perf.data file
  ftrace       simple wrapper for kernel's ftrace functionality
  inject       Filter to augment the events stream with additional information
  kallsyms     Searches running kernel for symbols
  kmem         Tool to trace/measure kernel memory properties
  kvm         Tool to trace/measure kvm guest os
  list         List all symbolic event types
  lock         Analyze lock events
  mem         Profile memory accesses
  record       Run a command and record its profile into perf.data
  report       Read perf.data (created by perf record) and display the profile
  sched        Tool to trace/measure scheduler properties (latencies)
  script       Read perf.data (created by perf record) and display trace output
  stat         Run a command and gather performance counter statistics
  test         Runs sanity tests.
  timechart   Tool to visualize total system behavior during a workload
  top          System profiling tool.
  version     display the version of perf binary
  probe       Define new dynamic tracepoints
  trace       strace inspired tool

See 'perf help COMMAND' for more information on a specific command.
```



# Profiling as a method for finding bottlenecks

Profiling is used to monitor the execution of a program to collect data on various aspects such as:

- execution time;
- resources used.

The purpose of profiling is to find bottlenecks or areas where the program can be optimized to improve its efficiency and performance.



# perf as a tool for analyzing software performance

perf is a profiling tool that is designed for Linux-based systems. Advantages:

- simple command line interface
- rich functionality.

To analyze the performance of SpdRoot we used such perf commands as:

- perf record
- perf report
- perf probe

```
[alxddd@ncx104 ~]$ perf
usage: perf [--version] [--help] [OPTIONS] COMMAND [ARGS]

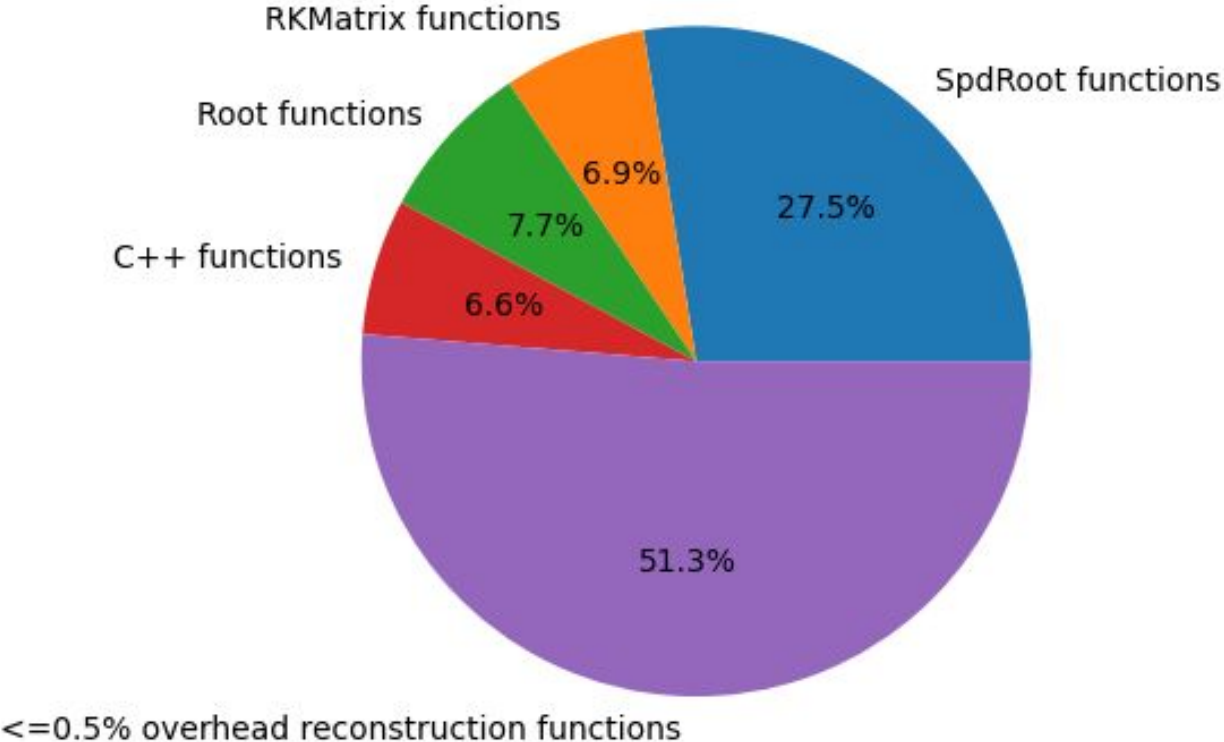
The most commonly used perf commands are:
  annotate      Read perf.data (created by perf record) and display annotated code
  archive      Create archive with object files with build-ids found in perf.data file
  bench        General framework for benchmark suites
  buildid-cache Manage build-id cache.
  buildid-list  List the buildids in a perf.data file
  c2c          Shared Data C2C/HITM Analyzer.
  config       Get and set variables in a configuration file.
  data        Data file related processing
  diff        Read perf.data files and display the differential profile
  evlist       List the event names in a perf.data file
  ftrace       simple wrapper for kernel's ftrace functionality
  inject       Filter to augment the events stream with additional information
  kallsyms     Searches running kernel for symbols
  kmem        Tool to trace/measure kernel memory properties
  kvm         Tool to trace/measure kvm guest os
  list        List all symbolic event types
  lock        Analyze lock events
  mem         Profile memory accesses
  record       Run a command and record its profile into perf.data
  report       Read perf.data (created by perf record) and display the profile
  sched       Tool to trace/measure scheduler properties (latencies)
  script      Read perf.data (created by perf record) and display trace output
  stat        Run a command and gather performance counter statistics
  test        Runs sanity tests.
  timechart   Tool to visualize total system behavior during a workload
  top         System profiling tool.
  version     display the version of perf binary
  probe       Define new dynamic tracepoints
  trace       strace inspired tool

See 'perf help COMMAND' for more information on a specific command.
```

## Characteristics in the case of a field of 1/8 of the total size

```
SpdFieldMap1_8 *MagField = new SpdFieldMap1_8("full_map");  
MagField->InitData("field_full1_8.bin");  
SpdRegion *reg = MagField->CreateFieldRegion("box");  
reg->SetBoxRegion(-330, 330, -330, 330, -386, 386); //  
(X,Y,Z)_(min,max), cm  
run->SetField(MagField);
```

# Characteristics in the case of a field of 1/8 of the total size. Resources used





## SpdRoot functions (27.5%) top CPU users

6.97% **genfit::RKTrackRep::RKPropagate** - old algorithm (2013) part of GENFIT. Extrapolation by Runge-Kutta method, many matrix operations.

5.60% **SpdFieldMap1\_8::Approx\_0** - function consists of multiplication and addition of vectors and matrices.

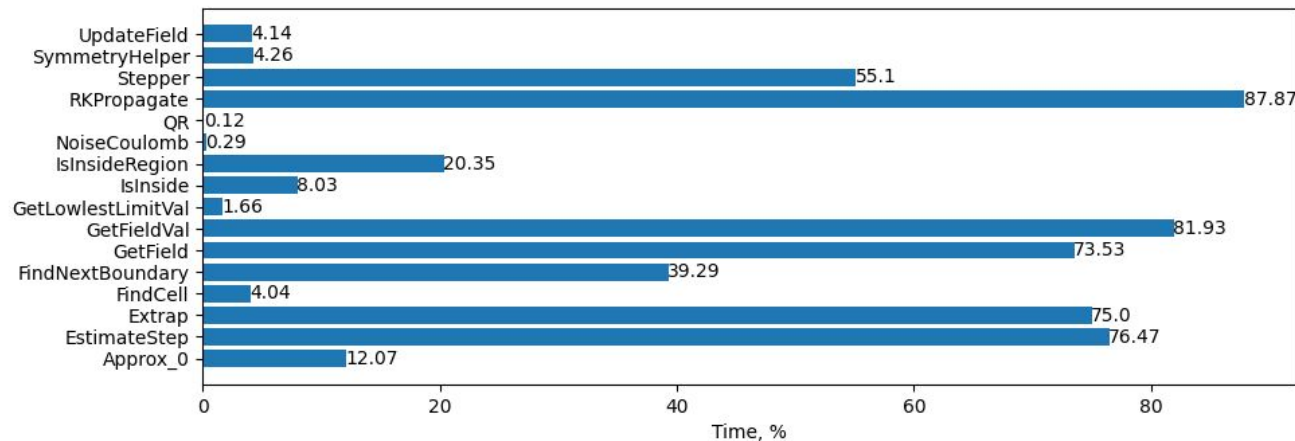
2.14% **SpdFieldMap1\_8::FindCell** - search for the cell in the field to which the point belongs.

1.61% **SpdFieldMap1\_8::GetField** - returns value of the field at the point.

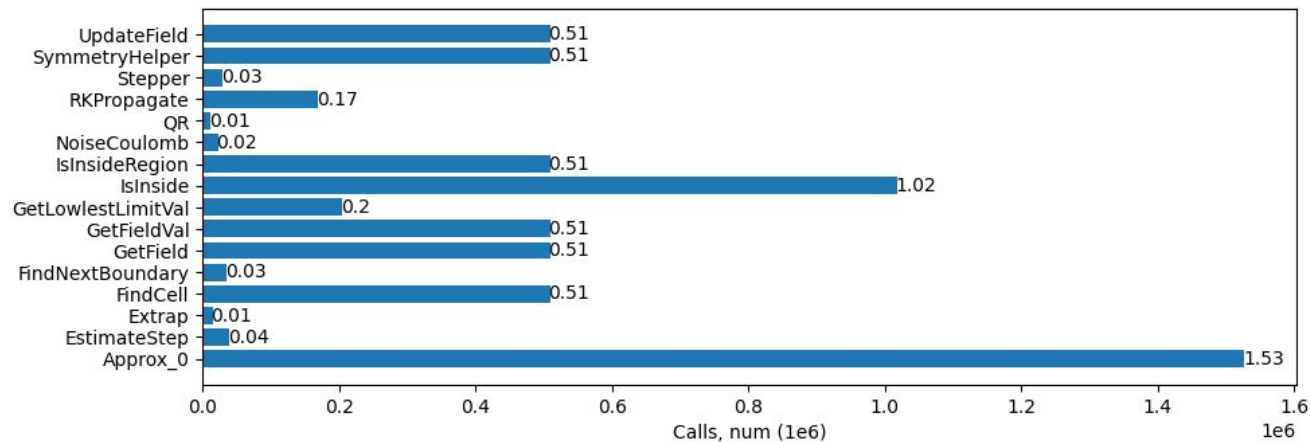
1.52% **SpdBoxRegion::IsInside** - check if the point is inside the area around the point (r,z).

# Characteristics in the case of a field of 1/8 of the total size.

## Execution time



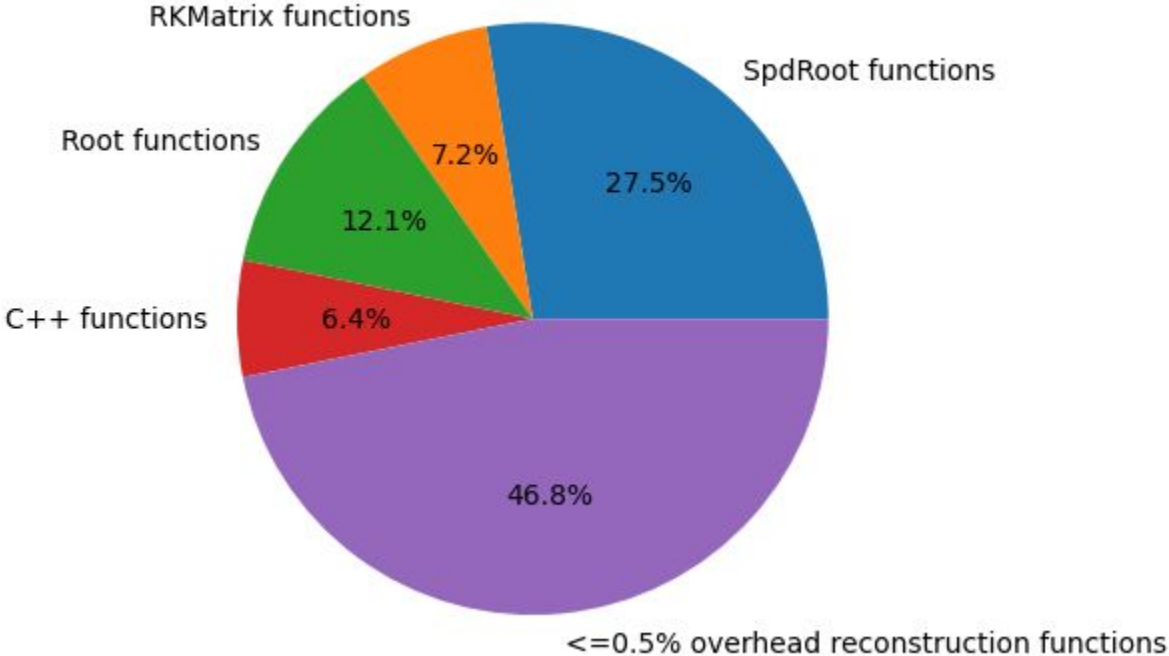
Total time = 124,0379  
sec per event  
(including perf)



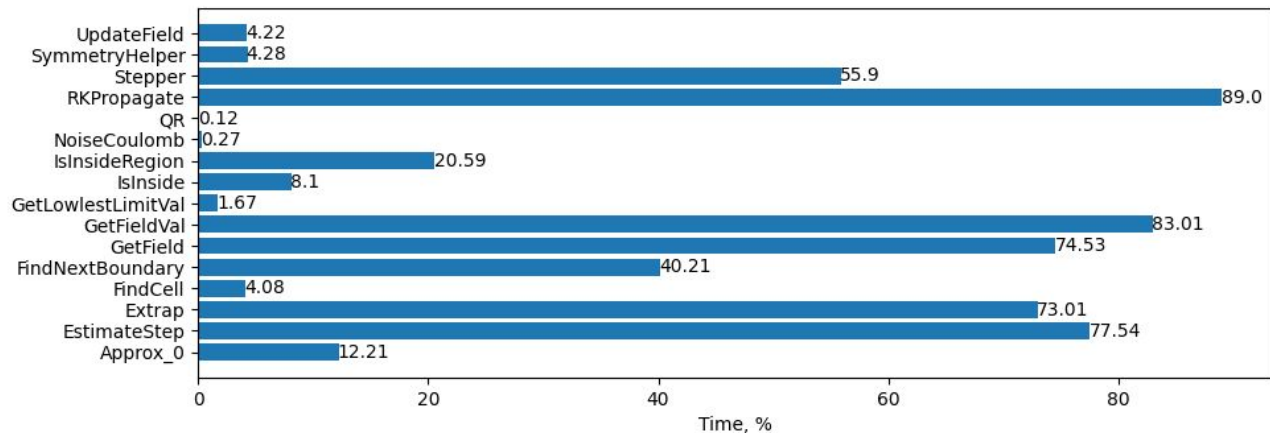
# Characteristics in case of constant field

```
SpdConstField* MagField = new SpdConstField();  
MagField->SetField(0., 0., 10.0); // kG  
SpdRegion* reg = 0;  
reg = MagField->CreateFieldRegion("tube");  
reg->SetTubeRegion(0, 174, -246, 246); // (R,Z)_(min,max),  
cm  
run->SetField(MagField);
```

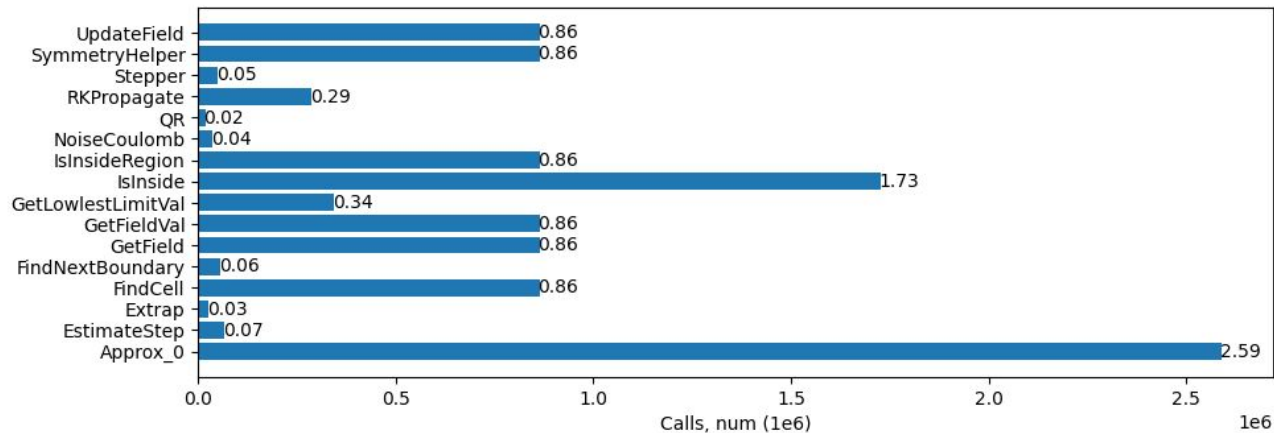
# Characteristics in case of constant field. Resources used



# Characteristics in case of constant field. Execution time



Total time = 205,3638 sec  
per event (including perf)



# Conclusion

- Different types of fields affect the reconstruction speed, but the percentages of function running time are quite close in different cases.
- There are differences in the number of function calls.
- Root - functions take more resources in the case of constant field.

## Plans:

- Get more statistics on more events and on other types of magnetic fields.
- Make sure that field parameters affect the reconstruction speed.

Thank you for your attention!