GRID’2025

10.07.2025

Search for bottlenecks in SpdRoot code

Didorenko Aleksei
didorenko@jinr.ru

Voytishin Nikolay
nvoytish@jinr.ru

MLIT JINR

Relevance

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

It is stated that the reconstruction runs slower than expected event processing speeds.
The current issue of SPD project is to detect bottlenecks in SpdRoot’s source code and further improve the processing speed and efficiency of computing resources.

Technology stack

```
[alxdid@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.
```

PVS-Studio

python™ pandas matplotlib

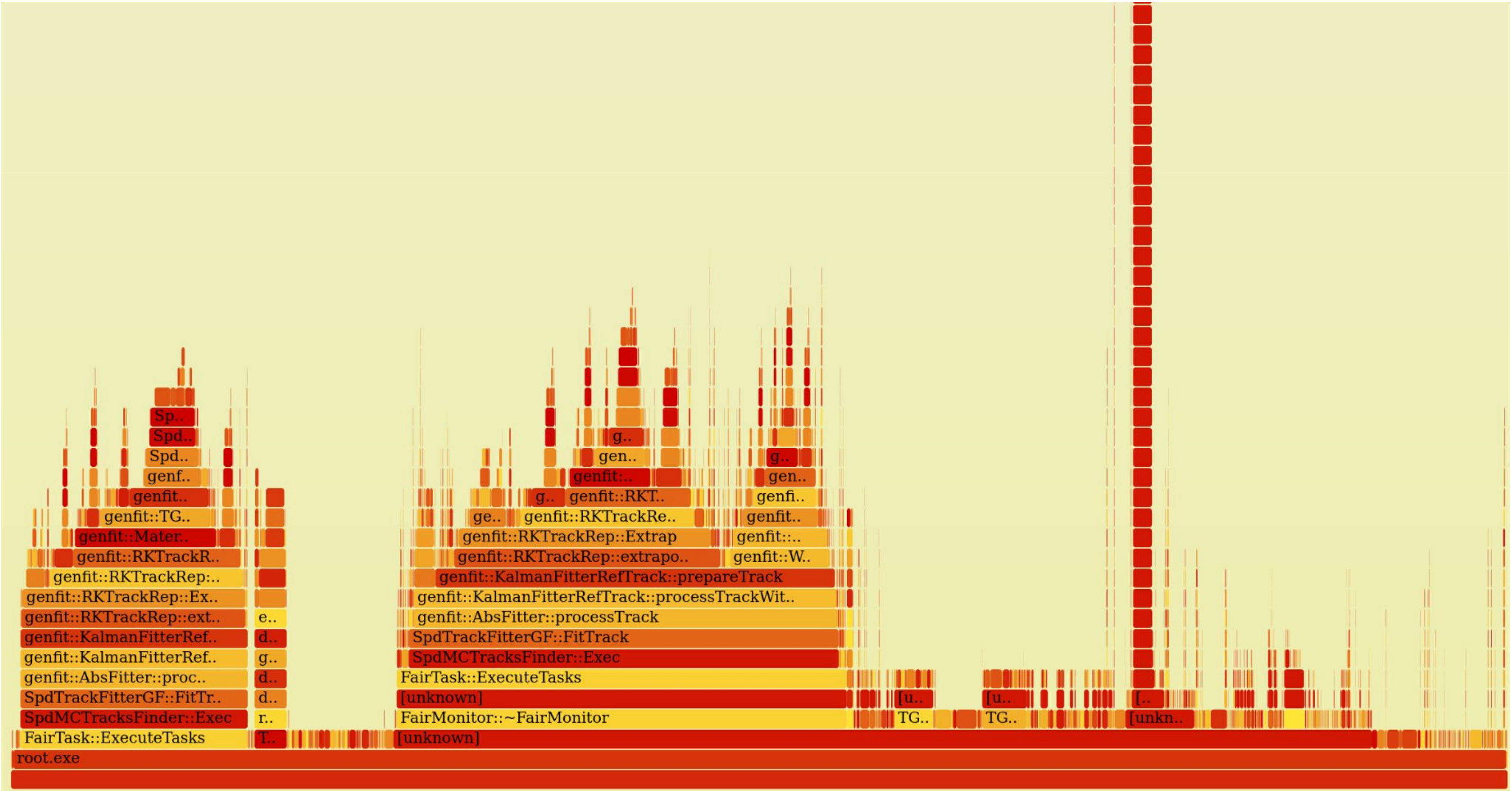
Profiling as a method for bottlenecks detection

Profiling is used to monitor the execution of a program to collect data on various aspects.
The purpose of profiling is to find bottlenecks or areas where the program can be optimized to improve its efficiency and performance.
Profiling can be:

- static
- dynamic

Reconstruction Flame Graph

FlameGraph displays the call of the reconstruction process functions. Each box - function; The width of the box shows the total time it was on-CPU; y-axis - stack depth; x-axis - sample population (sorted alphabetically); colors are random



PVS-Studio as a tool for static code profiling

PVS-Studio is a static analyzer of C, C++, C# and Java code designed to facilitate the task of finding and fixing various kinds of errors: Improper understanding of function/class operation logic, Incorrect handling of the types, Misprints, Dead code, Copy-Paste, Uninitialized variables, Unused variables, Undefined/unspecified behavior, etc.

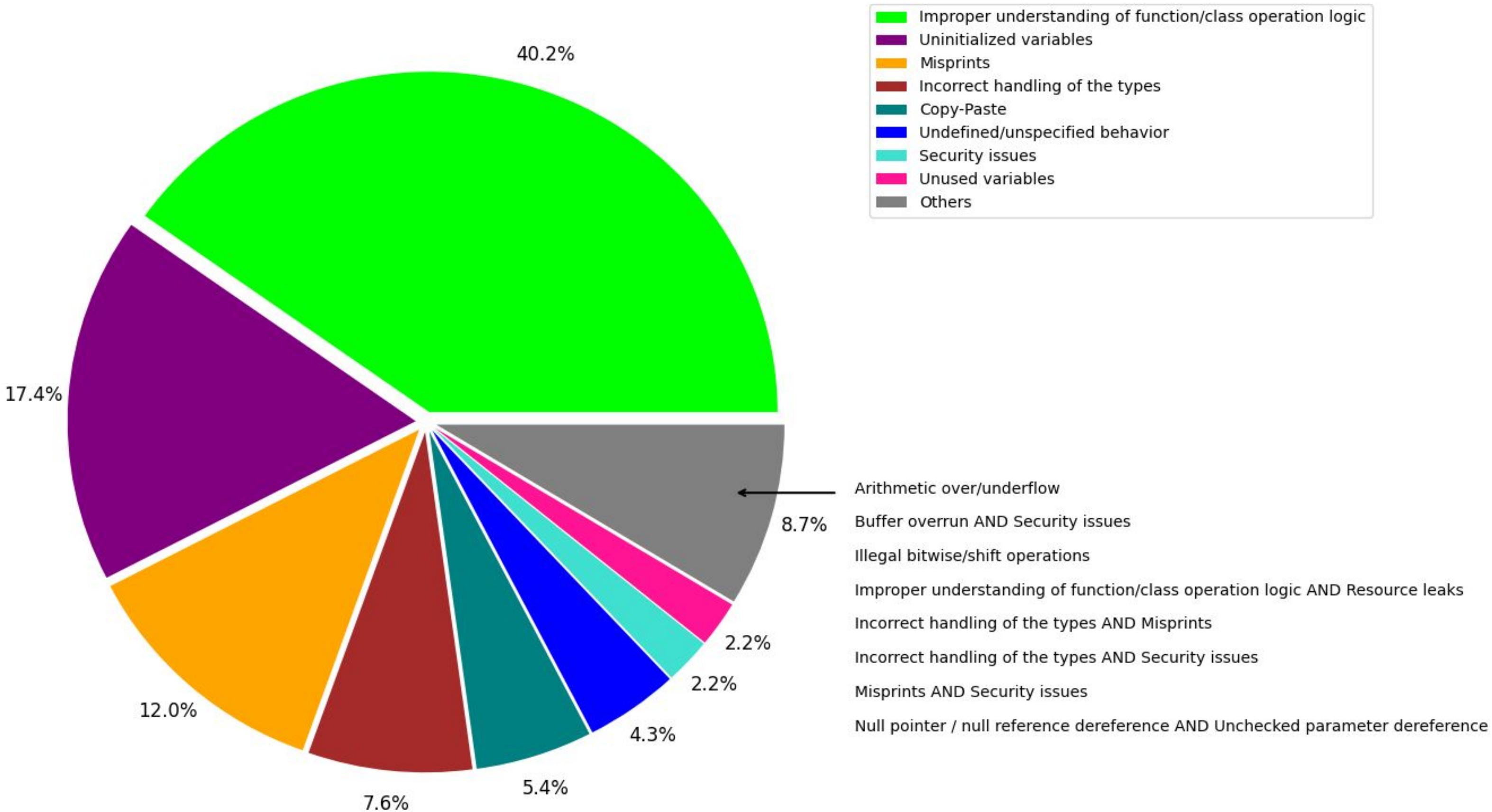
```
#Start analysis
pvs-studio-analyzer analyze -o /path/to/PVS-Studio.log \
                            -e /path/to/exclude-path \
                            -j<N>
#Get report
plog-converter -a GA:1,2 \
               -t json \
               -o /path/to/Analysis_Report.json \
               /path/to/PVS-Studio.log
```

Static code profiling results. JSON - report


```
{
    "code": "V678",
    "cwe": 688,
    "falseAlarm": false,
    "favorite": false,
    "level": 2,
    "message": "An object is used as an argument to its own method. Consider checking the first actual argument of the 'Transpose' function." ,
    "positions": [
      {
        "column": 1,
        "endColumn": 2147483647,
        "endLine": 966,
        "file": "/root/spdpvs/spdroot/external/GenFit2/trackReps/src/RKTrackRep.cc"  ,
        "line": 966,
        "navigation": {
          "columns": 0,
          "currentLine": 1153117847,
          "nextLine": 1988692485,
          "previousLine": 1391284327
        }
      }
    ],
    "projects": [],
    "sastId": ""
 }
```

Static code profiling results. Errors

The pie chart shows errors fraction of each type of the total number of errors. **Sections** – types of errors. **Percentages** - the percentage of error type from the total number of errors. **Gray section** contains types for which only one error was found (also errors that belong to several types at the same time).



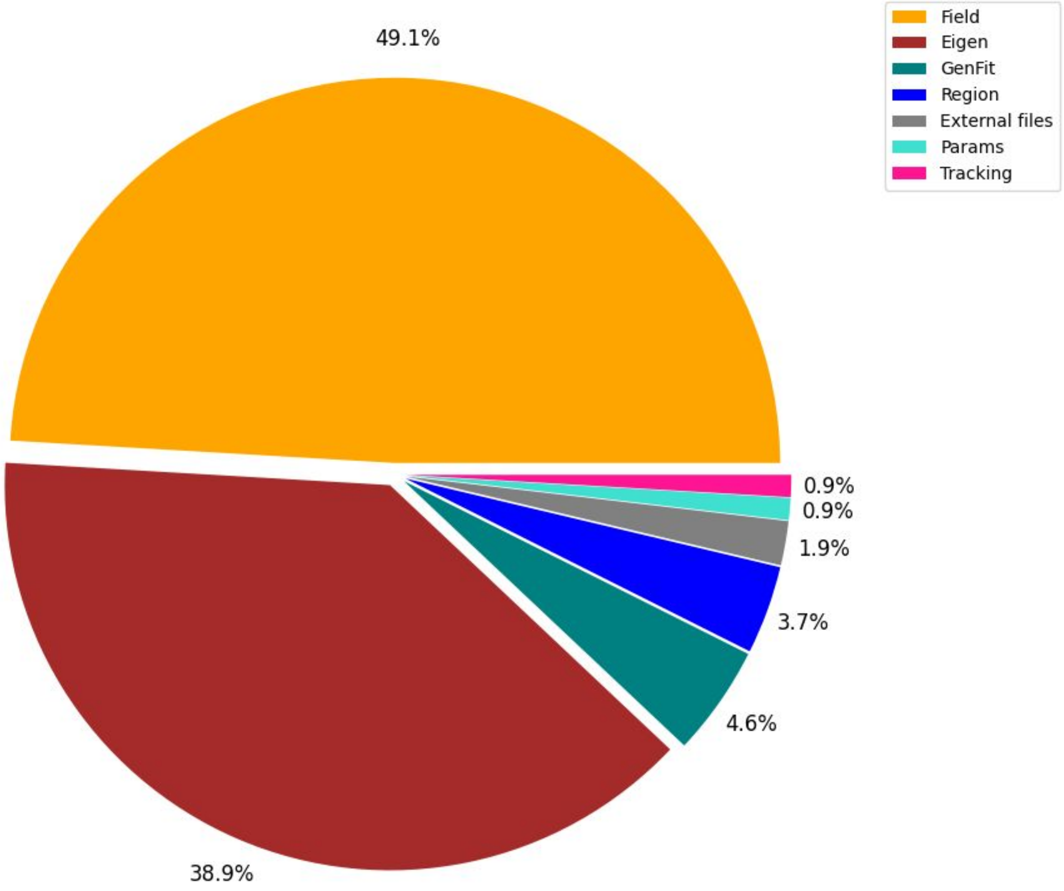
Static code profiling results. Files

The pie chart shows SpdRoot code parts with errors fractions from the total number of files with errors

Sections – part of the SpdRoot code.

Percentages - the percentage of SpdRoot code part with errors from the total number of files errors.

Gray section contains fraction of external files.



Non-void function should return a value

```
Bool_t
SpdBoxRegion::SetBoxRegion(Double_t xmin,
Double_t xmax, Double_t ymin,

Double_t ymax, Double_t zmin, Double_t
zmax)
{
  if ((xmax - xmin) < SR_EPS) {
fIsRegionInit = kFALSE; return kFALSE; }
  if ((ymax - ymin) < SR_EPS) {
fIsRegionInit = kFALSE; return kFALSE; }
  if ((zmax - zmin) < SR_EPS) {
fIsRegionInit = kFALSE; return kFALSE; }

  fXmin = xmin; fXmax = xmax;
  fYmin = ymin; fYmax = ymax;
  fZmin = zmin; fZmax = zmax;

  fIsRegionInit = kTRUE;
  return kTRUE;
}
```

```
Bool_t
SpdTubeRegion::SetTubeRegion(Double_t
rmin, Double_t rmax,

Double_t zmin, Double_t zmax)
{
   if ((rmax - rmin) < SR_EPS) {
fIsRegionInit = kFALSE; return kFALSE; }
   if ((zmax - zmin) < SR_EPS) {
fIsRegionInit = kFALSE; return kFALSE; }

   fRmin = rmin; fRmax = rmax;
   fZmin = zmin; fZmax = zmax;

   fIsRegionInit = kTRUE;
   return kTRUE;
}
```

```
Bool_t
SpdPhysicalRegion::SetPhysicalRegion(TStr
ing region, TString type)
{
   fRegion = region;

   fType = -1;

   if (type == "volume") fType = 0;
   else if (type == "medium") fType = 1;

   fIsRegionInit = kTRUE;
   return kTRUE;
}
```

Possible overflow. Consider casting operands of the 'xi * twoopi_1' operator to the 'uint64_t' type, not the result

```
p = (uint64_t(xi * twoopi_1) << 32) + p;
```

```
p = ((uint64_t(xi) * uint64_t(twoopi_1)) << 32) + p;
```

Alignment and data loading errors

```
template<> EIGEN_STRONG_INLINE Packet4f ploaddup <Packet4f>(const  float*   from)
{
  return vec4f_swizzle1 (_mm_castpd_ps (_mm_load_sd (reinterpret_cast<const  double*>(from))), 0, 0, 1, 1);
}
```

```
template<> EIGEN_STRONG_INLINE Packet4f ploaddup <Packet4f>(const  float* from)
{
    m128i tmp  =  mm_loadl_epi64(reinterpret_cast<const    m128i*>(from));
  return vec4f_swizzle1 (_mm_castsi128_ps(tmp), 0, 0, 1, 1);
}
```

Suspicious implicit type casting

```
template<> EIGEN_STRONG_INLINE Packet4f pblend (const Selector<4>& ifPacket, const Packet4f& thenPacket,
const Packet4f& elsePacket) {
  const __m128 zero = _mm_setzero_ps();
  const __m128 select = _mm_set_ps(ifPacket.select[3] ? 1.0f : 0.0f, ifPacket.select[2] ? 1.0f : 0.0f,
ifPacket.select[1] ? 1.0f : 0.0f, ifPacket.select[0] ? 1.0f : 0.0f);
  __m128 false_mask = _mm_cmpeq_ps(select, zero);
#ifdef EIGEN_VECTORIZE_SSE4_1
  return _mm_blendv_ps(thenPacket, elsePacket, false_mask);
#else
  return _mm_or_ps(_mm_andnot_ps(false_mask, thenPacket), _mm_and_ps(false_mask, elsePacket));
#endif
}


template<> EIGEN_STRONG_INLINE Packet2d pblend (const Selector<2>& ifPacket, const Packet2d& thenPacket,
const Packet2d& elsePacket) {
  const __m128d zero = _mm_setzero_pd();
  const __m128d select = _mm_set_pd(ifPacket.select[1] ? 1.0 : 0.0, ifPacket.select[0] ? 1.0 : 0.0);
  __m128d false_mask = _mm_cmpeq_pd(select, zero);
#ifdef EIGEN_VECTORIZE_SSE4_1
  return _mm_blendv_pd(thenPacket, elsePacket, false_mask);
#else
  return _mm_or_pd(_mm_andnot_pd(false_mask, thenPacket), _mm_and_pd(false_mask, elsePacket));
#endif
}
```

Possible realloc() leak

```
inline void* handmade_aligned_realloc(void* ptr, std::size_t size, std::size_t = 0)
{
  if (ptr == 0) return handmade_aligned_malloc(size);
  void *original = *(reinterpret_cast<void**>(ptr) - 1);
  std::ptrdiff_t previous_offset = static_cast<char *>(ptr)-static_cast<char
*>(original);
  void* temp = std::realloc(original, size + EIGEN_DEFAULT_ALIGN_BYTES);
  if (temp == nullptr) return nullptr;
  original = temp;
  if (original == 0) return 0;
  void *aligned = reinterpret_cast<void*>((reinterpret_cast<std::size_t>(original) &
~(std::size_t(EIGEN_DEFAULT_ALIGN_BYTES-1))) + EIGEN_DEFAULT_ALIGN_BYTES);
  void *previous_aligned = static_cast<char *>(original)+previous_offset;
  if (aligned != previous_aligned)
      std::memmove(aligned, previous_aligned, size);

  *(reinterpret_cast<void**>(aligned) - 1) = original;
  return aligned;
}
```

The 'StateOnPlane' class implements the copy assignment operator, but lacks a copy constructor. It is dangerous to use such a class.

```
StateOnPlane(const StateOnPlane& other)
      : state_(other.state_),
        auxInfo_(other.auxInfo_),
        sharedPlane_(other.sharedPlane_),
        rep_(other.rep_)
  {}
```

bField[2]' variable is assigned values twice successively

```
if (v < RTOLERANCE) {
      // phi = pi, cos(phi) = -1, sin(phi) = 0
      // (see ROOT TVector2 phi angle definition for x = y = 0)
      //cosp = -1;
      //sinp =  0.;
      switch (fApproxMethod)
      {
         case 0 : {bField[2] = Approx_0(fFz); break;}
         case 1 : {bField[2] = Approx_1(fFz); break;}
         case 2 : {bField[2] = Approx_2(fFz); break;}
      }

      bField[0] = 0;
      bField[1] = 0;

      return kTRUE;
 }
```

The 'fApproxMethod' variable is assigned values twice successively

```
void
SpdAxialFieldMap::SetApproxMethod(Int_t
method)
{
   if (method < 0 || method > 2) {
       cout <<
"<SpdAxialFieldMap::SetApproxMethod> "
           << " Unknown method
approximation type: " << method
           << ", set default (0) " <<
endl;
       fApproxMethod = 0;
   } else {
       fApproxMethod = method;
   }
}
```

```
void SpdFieldMap::SetApproxMethod(Int_t
method)
{
   if (method < 0 || method > 2) {
       cout <<
"<SpdFieldMap::SetApproxMethod> "
           << " Unknown method
approximation type: " << method
           << ", set default (0) " <<
endl;
       fApproxMethod = 0;
   } else {
       fApproxMethod = method;
   }
}
```

```
void
SpdFieldMap1_8::SetApproxMethod(Int_
t method)
{
   if (method < 0 || method > 2) {
       cout <<
"<SpdFieldMap1_8::SetApproxMethod> "
           << " Unknown method
approximation type: " << method
           << ", set default (0) "
<< endl;
       fApproxMethod = 0;
   } else {
       fApproxMethod = method;
   }
}
```

Reconstruction startup parameters

The simulation and reconstruction were run using the example of the decay of a j-psi particle into two muons

NICA / spdroot

master

spdroot / macro / examples / jpsi-mumu

Updated jpsi-mumu example
Igor Denisenko authored 1 year ago

Name	Last commit
..	
analyze_jpsi.C	Updated jpsi-mumu example
fit_dimu.C	upadate 270321
reco.C	Updated jpsi-mumu example
run_all.sh	Updated jpsi-mumu example
simu.C	Updated jpsi-mumu example

Original version of SpdRoot generated a dataset with 1000 events.
Reconstruction process was executed 5 times in every versions.

The magnetic field is 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);
```

Results

	Original version	Our version
Real time, s	4,56301	4,50158
CPU time, s	4,56253	4,5011
reco.root file size, byte	572912137	572912137

For 1 event

For 1000 events

Conclusion

- Key issues detected in the SpdRoot code include: integer overflow risks, unsafe type casting, memory leaks, missing copy constructors, and logical errors
- Proposed fixes slightly reduced average reconstruction time per event without compromising output integrity.

The fixed syntax issues take into account the recommendations of the static code analyzer. The problems with the logic were referred to the developers and those responsible for the source code of SpdRoot.

Thank you for your attention!